Escola Politècnica Superior
de Castelldefels

UNIVERSITAT POLITÈCNICA DE CATALUNYA

TREBALL DE FI DE CARRERA

TÍTOL DEL TFC: Introducció als sistemes digitals reconfigurables amb la targeta DE2 d'Altera

TITULACIÓ: Enginyeria Tècnica de Telecomunicació, especialitat Sistemes de Telecomunicació

AUTOR: **Roser Ortiz Ramos**

DIRECTOR: Francesc Josep Sànchez i Robert

DATA: 18 de març de 2009

Títol: Introducció als sistemes digitals reconfigurables amb la targeta DE2 d'Altera

Autor: Roser Ortiz Ramos

Director: Francesc Josep Sànchez i Robert

Data: 18 de març de 2009

Resum

Amb aquest treball volem fer una petita introducció a la placa DE2 d'Altera. Veurem una introducció als dispositius que aquesta placa posa al nostre abast i a les FPGA's.

Per començar a posar-la en marxa utilitzarem el programa Quartus II, d'Altera, amb el que podem programar la FPGA en diferents llenguatges. Nosaltres ens decantarem pel VHDL. Començarem fent aplicacions senzilles que anirem complicant poc a poc afegint dispositius. Finalment crearem un rellotge digital amb diferents mòduls que prèviament haurem compilat i simulat individualment.

A la tercera part del projecte crearem el nostre propi microcontrolador NIOS. Aquesta possibilitat que ens dona Altera amb el seu NIOS II ens facilitarà la feina. Ho veurem creant petites aplicacions i anirem complicant-les fins arribar a crear el nostre propi rellotge digital, aquest cop programat amb llenguatge C. En aquesta part també utilitzarem el Quartus II, aquest cop amb l'aplicació SOPC Builder. Per carregar el nostre projecte en C a la placa utilitzarem l'Altera Monitor Program.

Finalment veurem les diferències principals que em trobat entre les dues maneres de treballar.

Title: Introduction to digital systems with reconfigurable card for Altera DE2

Author: Roser Ortiz Ramos

Director: Francesc Josep Sànchez i Robert

Date: March, 18th 2009

Overview

With this work we want to do a short introduction to Altera DE2 board. We will see the different devices that Altera offers in this board and the definition of FPGA.

We start programming the FPGA with Quartus II, of Altera. We can program this device with different languages. We will choose VHDL. We will start with simple applications and we will be more complicated. Finally, we will do a digital clock with different modules that we have compiled and simulated previously.

In the third of this project we will create our microcontroller NIOS. This possibility offered by the Altera NIOS II does easier the work. We will program small applications and we complicate them up to create our digital clock, this programmed with C. In this point we will use the Quartus II program with the application SOPC Builder. To load the C file to the board, we will use the Altera Monitor Program.

Finally we see the main differences I found between two ways of working.

ÍNDEX

INTRODUCCIÓ

El propòsit d'aquest projecte és posar en marxa la placa DE2 d'Altera. Estudiar les possibilitats que ens ofereix i fer un dossier on puguem veure fàcilment el funcionament bàsic d'aquesta placa.

En la primera part farem una introducció a la pròpia placa i a les FPGA's, xip que programarem més endavant. Amb aquesta introducció volem que el lector sàpiga en tot moment sobre el que estem treballant i les possibilitats que ens ofereix.

Un cop sabem que és DE2 començarem a crear petites aplicacions i anirem posant en marxa, poc a poc, diferents dispositius. Tot això ho farem amb el programa Quartus II d'Altera, creat per a la programació de FPGA's. Veurem com crear nous projectes, compilar-los, simular-los i carregar-los a la placa. Aquesta segona part acabarà amb la realització d'un projecte, un rellotge digital, on aplicarem tots els coneixements adquirits a els apartats anteriors on haurem fet petits projectes per posar en marxa els interruptors, els leds, botons i els displays 7 segments.

En una tercera part aprendrem a construir el nostre propi microcontrolador per així estalviar-nos feina a l'hora de programar els dispositius. Veurem com els programes que utilitzem, SOPC Builder juntament amb el Quartus s'encarreguen de generar-nos els arxiu que carregarem a la placa per indicar-li quin és el nostre NIOS. Programarem el dispositiu amb llenguatge C fent una petita primera referència al llenguatge assemblador. Anirem poc a poc, creant aplicacions petites, començant per una senzilla assignació d'entrada sortida que anirem millorant fins arribar a crear el nostre propi rellotge digital. Utilitzarem i manipularem interrupcions i posarem en marxa interruptors i botons que ens ofereix la placa; la sortida la podrem visualitzar als displays 7 segments.

Arribats aquest punt farem una petita observació de les diferències entre el rellotge creat en VHDL i el rellotge creat en C.

1. Introducció a la placa DE2 d'Altera

1.1. Què és Altera?

Altera és la companyia pionera en dispositius programables. Des de que va començar aquesta industria, al 1983, Altera ha anat guanyant en qualitat, facilitats i, el més important per una empresa, en capacitat de mercat.
Aquesta empresa ens ofereix un gran ventall de productes utilitzables per a diferents aplicacions. Podem trobar solucions per l'automoció, medicina, industries, i fins i tot per fins militars.

Els productes que ens ofereix són, entre d'altres: FPGAs, CPLDs i ASICs, tots aquest de diferents qualitats i preus per adaptar-se a les necessitats de l'usuari.
A més de l'ampli catàleg de productes que ens ofereix Altera, ells mateixos, ens proporcionen les eines suficients per poder posar en marxa i utilitzar els seus dispositius. A la seva pàgina web [1] podem trobar des del software necessari, fins a tutories i programes preparats per a que l'usuari no es perdi en la utilització. Tot ho ofereix de forma gratuïta, tot i que has d'estar registrat a la seva web per poder obtenir la llicència pel programari.

1.2. Què es una FPGA?

Field Programmable Gate Array. Es tracta d'un dispositiu semiconductor reprogramable. Té la capacitat de fer des de càlculs senzills com ara simular una porta lògica fins a aplicacions complexes.

Està constituïda per diferents blocs lògics que el usuari programa mitjançant el llenguatge i software adient.

A més d'Altera hi ha altres fabricants com ara: *Xilinx, Lattice Semiconductor, Achronix Semiconductor.*

1.3. Què és DE2?

Un dels productes més innovadors és la placa DE2. A més d'oferir-nos la FPGA ens incorpora en una mateixa placa una gran varietat de dispositius i entrades i sortides, totes encapsulades en una mateixa placa.
Aquesta és la DE2 on es combina una FPGA d'Altera amb un processador NIOS II. A més d'aquest dispositiu programable també presenta diferents connectors entrada / sortida i dispositius de visualització de dades com ara una

petita pantalla LCD, diferents LEDs i displays. Ho veiem més especificat a la taula que ens ofereix el fabricant [1]:

'FPGA

- *Cyclone II EP2C35F672C6 with EPCS16 16-Mbit serial configuration device*

I/O Devices

- *Built-in USB-BlasterTM cable for FPGA configuration*
- *10/100 Ethernet*
- *RS232*
- *Video Out (VGA 10-bit DAC)*
- *Video In (NTSC/PAL/Multi-format)*
- *USB 2.0 (type A and type B)*
- *PS/2 mouse or keyboard port*
- *Line In/Out, Microphone In (24-bit Audio CODEC)*
- *Expansion headers (76 signal pins)*
- *Infrared port*

Memory

- *8-MBytes SDRAM, 512K SRAM, 4-MBytes Flash*
- *SD memory card slot*

Displays

- *16 x 2 LCD display*
- *Eight 7-segment displays*

Switches and LEDs

- *18 toggle switches*
- *18 red LEDs*
- *9 green LEDs*
- *Four debounced pushbutton switches*

Clocks

- *50 MHz crystal for FPGA clock input*
- *27 MHz crystal for video applications*
- *External SMA clock input'*

Més endavant utilitzarem una part d'aquest dispositius en diferents aplicacions. Veiem una fotografia de la placa on diferenciem tots aquests:

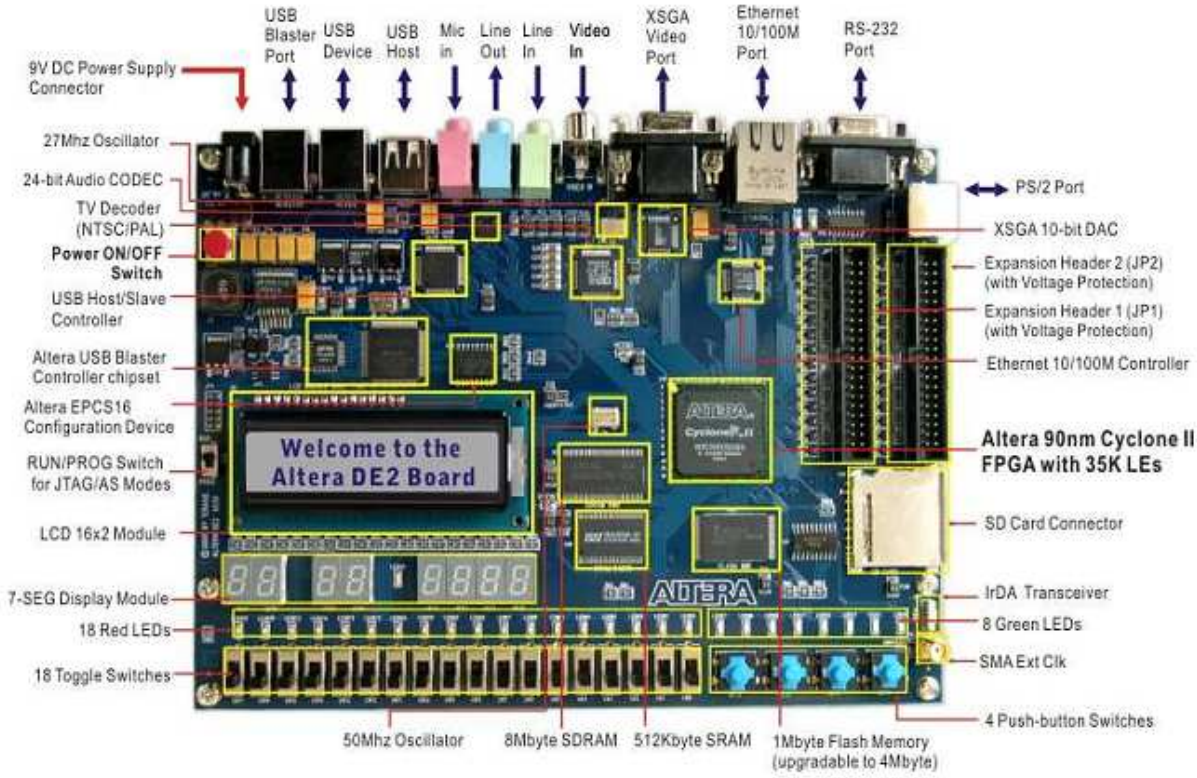


Fig. 1.1. Fotografia de la placa DE2

La placa DE2 està pensada per l'educació. Com em vist tenim molts dispositius a l'abast d'una mateixa placa amb la que podem fer gran quantitat de projectes com els que el fabricant proposa a les seves tutories. Des d'una assignació senzilla d'entrada/sortida a un programa més complex com ara una màquina de *karaoke*.

Això es pot fer optimitzant el cost computacional de la FPGA gràcies a que nosaltres mateixos ens podem crear el nostre processador. Altera ens ofereix un dispositiu que podem modelar mitjançant un software específic per tal d'utilitzar només la memòria justa per a que els nostres projectes no acumulin retards innecessaris per dispositius que no s'utilitzen.

1.4. Utilització: Programació i eines

1.4.1. Programació de la FPGA

Com he dit anteriorment el propi fabricant, Altera, ens proporciona les eines per poder treballar amb la placa.

Programació de la FPGA

Tenim la possibilitat d'utilitzar dos llenguatges de programació diferents: *Verilog* i *VHDL*. En aquest cas l'elecció és *VHDL*, llenguatge utilitzat a les classes de l'assignatura de *SED*. A més, el que fa més manejable aquest llenguatge en lloc de *Verilog* és el sistema de llibreries i de mòduls que es pot aplicar als projectes. És a dir, podem anar dissenyant per mòduls més senzills un projecte de gran dimensions; provar el funcionament individual i, finalment, incorporar tots aquests dins d'un mateix projecte. La compilació es farà de manera conjunta com si tot es tractés d'un mateix mòdul sense la necessitat de refer-ho tot en un.

Pel que fa els manuals i exercicis que ens proposa Altera, tenim possibilitat d'escollir entre tots dos programaris, tenim els mateixos exercicis a la seva web [1] tant el per un com per l'altre.

VHDL és un llenguatge molt sistemàtic i amb una estructura fixa que es repeteix a tots els sistemes:

- *Llibreries:* El que fa que el programa pugui compilar diferents mòduls dins d'un mateix projecte.
- *Entity:* Part del projecte on indiquem les entrades i sortides que té el sistema global.
- *Architecture*: És el cos del projecte, on declarem senyals internes i les funcions que el sistema ha de seguir.

Veiem un exemple senzill d'un projecte en que es dona un valor d'entrada a la sortida directe:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;

ENTITY SWiLEDS IS
      PORT( SW    :  IN STD_LOGIC_VECTOR(17 DOWNTO 0);
            LEDR  :  OUT STD_LOGIC_VECTOR(17 DOWNTO 0)
            );
END SWiLEDS;

ARCHITECTURE Behavior OF SWiLEDS IS
   BEGIN
```

```
        LEDR <=SW;
    END Behavior;
```

Veurem més exemples en VHDL al llarg del tutorial i comprovarem com aquesta estructura sempre es compleix.

1.4.2. Creació i programació del NIOS

Pel que fa referència al NIOS tenim dos parts diferenciades que programar. Podríem dir que una és el Hardware del processador o arquitectura interna i una altra el software o aplicació que volem que executi el NIOS.

Disseny de Hardware

Veurem que per dissenyar el nostre processador l'eina és molt visual i no necessitem tindre grans coneixement de cap llenguatge de programació. El propi programa que utilitzarem per aquest fi ens crearà un codi per poder-ho carregar al Quartus i així traslladar-ho a la placa. Per una altra banda, sí que podem escollir en quin llenguatge es crearà dit fitxer, en VHDL o Verilog.

Disseny del Software

Un cop tenim el disseny especificat sí que tenim que decantar-nos per el llenguatge en assemblador o en C. En el nostre cas sempre serà més fàcil, per la nostra trajectòria acadèmica, utilitzar el llenguatge C. En canvi l'assemblador és un llenguatge molt específic per cada dispositiu, amb unes instrucció molt determinades i definides. A més, com el nostre processador no és fix, el dissenyem nosaltres mateixos, el manual que podem tindre no té perquè estar adaptat al cent per cert al nostre dispositiu. Per poder entendre i aprendre a treballar amb aquest llenguatge de programació Altera ens proporciona un manual [4][5] on s'expliquen cada una de les instrucció, estructura i funcionalitat de manera genèrica. Sempre tindrem funcionalitats d'ajuda als programes que utilitzem per fer consultes de les instruccions que es podran utilitzar.

Utilitzarem cada un d'aquest llenguatges en petits exemples. Posteriorment, farem un projecte més elaborat utilitzant el llenguatge C.

1.4.3. Eines

Al CD que ens incorporen a la mateixa capça de la placa o a la web d'Altera trobem els diferents programes que utilitzarem per a programar la nostra placa.

Tenim diferents programes com ara el *Quartus II, SOPC Builder*, *Aletra Monitor Program.*

Quartus II:

Aquest software l'utilitzarem per crear aplicacions, simular-les i carregar-les a la placa. Altera ens deixa adquirir una edició gratuïta des de la seva web. Per la versió que ens ofereixen al CD que s'inclou amb la placa, v7.1, es requereix una llicència, gratuïta, que ens permet utilitzar el programa per aprendre el funcionament. Per una altra banda, podem obtenir una última versió, la v8.1 per la qual aquesta llicència ja no és requerida.

A més a més tenim la versió de pagament. Per saber quina és la millor per el nostre treball veiem les diferencies detalladament.

Web Edition vs Subscrition Edition:

Principalment, des d'un punt de vista econòmic la elecció serà la edició Web degut a que és gratïuta i les prestacions són amplies.

El sistema operatiu que utilitzem ens pot fer variar entre una edició o una altra ja que la edició Web només es compatible amb Widows Vista o XP (32 bit). En canvi si el sistema utilitzat és Linux o Windows Vista o XP de 64 bits tenim que escollir la edició amb subscripció.

Pel que fa a xips programables amb aquestes eïnes, la edició de subscripció ens aumenta les posibilitats. Pel que fa a ASIC l'edició web no ens dona la possibilitat de programar-les.

Respecte a les IP disponibles amb l'edició gratuïta, ens proporcionen la eina de *OpenCore Plus Evaluation* per tots els dispositius Altera. Si es vol la llicència completa per IP, es pot demanar i comprar. A l'edició de pagament, venen totes les IPs incloses, tant per DSP com per a *Memory Controllers*.

Com a estudiants no ens fa falta tindre cap més IP, amb el OpenCore poden crear, simular, carregar i verificar el nostre projecte a la placa. Tot això de manera ràpida i fàcil. La llicència per la resta de IP's està pensada per aquell moment en que el nostre projecte estigui llest per enviar-ho a producció; sempre i quan tingui els requeriments i funcionalitats adients per fer-ho.

Per la resta de prestacions com ara rapidesa, optimització i verificacions dels dissenys, totes dues edicions donen els mateixos resultats. Altera a la seva pàgina ens ofereix una taula amb totes les diferències entre una i l'altre.

Un cop feta l'elecció:

Aquest és el primer programa que posarem en marxa, el Quartus II. Per poder utilitzar-ho, segons la versió utilitzada, tenim que tindre la llicència d'Altera que podem sol·licitar a través de la web [1] responent a un seguit de preguntes. Aquesta llicència és totalment gratuïta.

La instal·lació del programa no és complicada, s'obre un diàleg d'instal·lació típic de Windows:

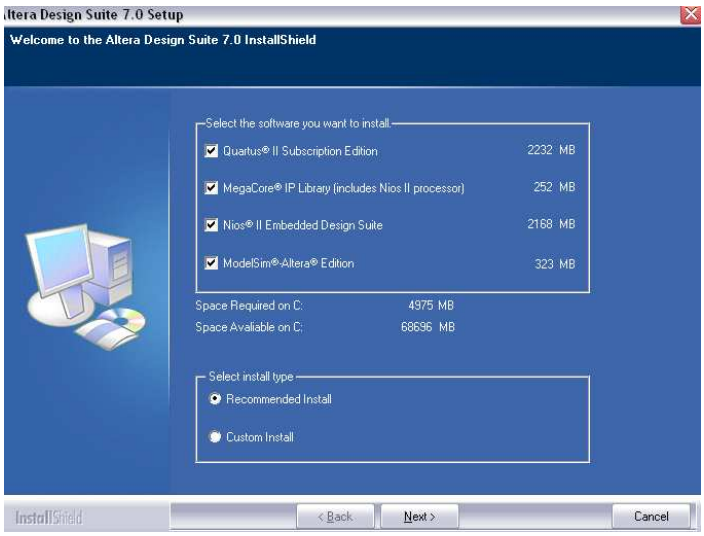


Fig. 1.2. Diàleg d'instal·lació de Windows

Durant la instal·lació no ens demanen la llicència, ho farà el programa la primera vegada que l'executem. Un cop es valida que la llicència és correcte tenim el programa llest per començar la nostra feina. La pantalla principal del Quartus II sobre la que treballarem serà la següent:

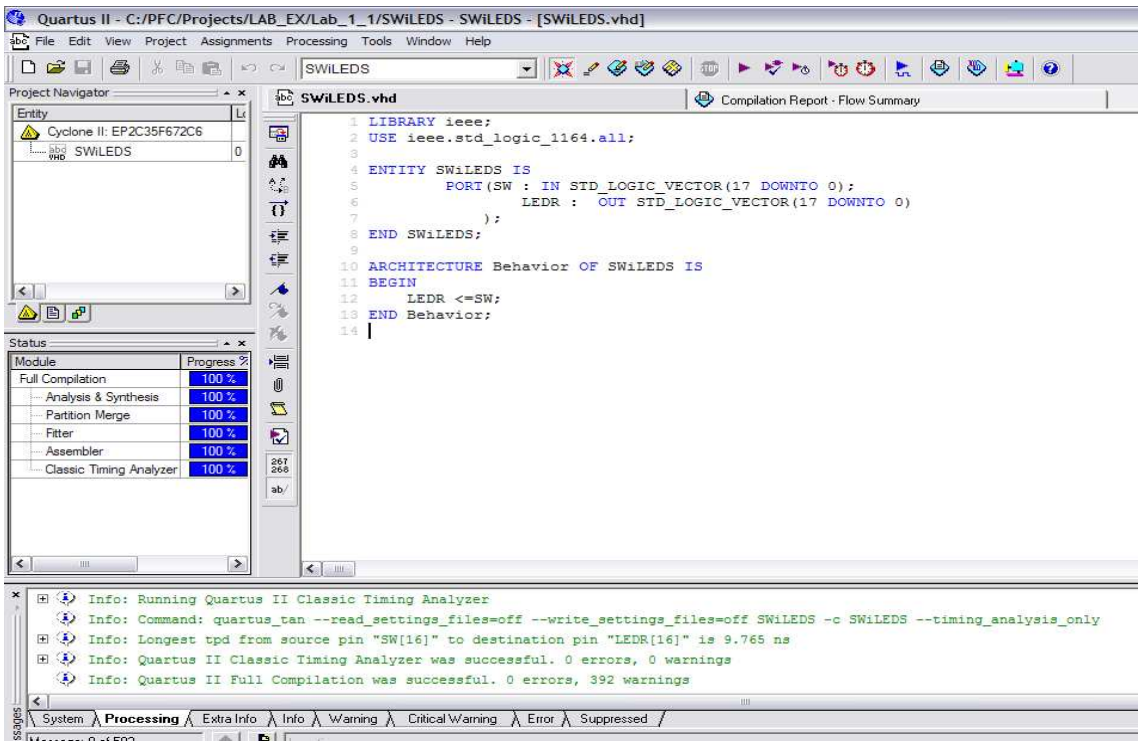


Fig. 1.3.Pantalla principal de Quartus II

A la part esquerra de la pantalla trobem la informació del projecte com ara: dispositiu que estem utilitzant, nom del projecte i a la pestanya '*files'*, els arxius que el composen. A la part central veiem diferents pestanyes on es fa el treball principal com editar els nostres arxius o veure informació que ens dona el Quartus sobre la compilació. A la part inferior ens apareixeran els *Warnings* i errors que ha trobat a l'hora de compilar el projecte. A la part superior veiem les principals eines per poder treballar (compilar, simular, etc).

SOPC Builder

Un cop instal·lem el Quartus II tenim aquest programa també instal·lat. SOPC Builder ens permet construir el nostre propi NIOS partint d'uns dispositius que té enllistats. A més cada un d'aquest el podem manipular, sempre dintre de les possibilitats de cada un. Per accedir a SOPC Builder tenim que tindre el Quartus II funcionat ja que és necessari un projecte *.qpf*.

Podem accedir a SOPC Builder un cop obert un projecte al Quartus pel menú:

Tools → SOPC Builder

Un cop obert veurem la pantalla principal:

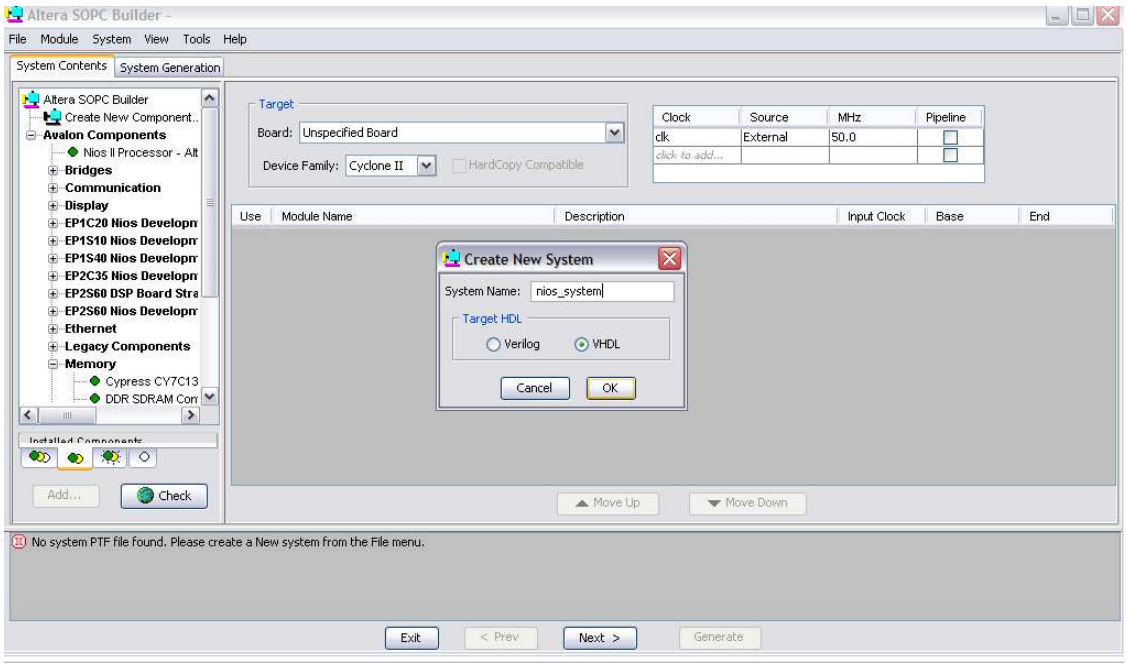


Fig. 1.4. Pantalla principal del SOPC Builder.

Quan indiquem que volem construir el NIOS, el programa s'encarrega de generar tots els arxius per a que tant el Quartus com l'Altera Monitor Program entenguin l'arquitectura interna d'aquest microcontrolador fet a mida. Per aquesta raó a la primera pantalla li tenim que indicar el nom de l'arxiu que després incorporarem al projecte de l'Altera Monitor i el llenguatge dels arxius que adjuntarem al Quartus per posteriorment fer l'assignació de Pins i així establir comunicació entre la placa i el nostre programa.

A la part esquerra de la pantalla principal del SOPC (Fig. 1.4) tenim el llistat de tots els dispositius que Altera posa a la nostra disposició per la nostra placa. A la part superior de la pantalla veiem els identificadors de la nostra placa: Família a la que pertany i el nom de la pròpia placa.

Altera ens proporciona un manual on ens ajuden a començar a utilitzar aquest programa amb una petita aplicació. Aquest manual el trobem a la web d'Altera [1] i el podem baixar de manera gratuïta.

Veurem amb més profunditat el funcionament d'aquest programa en apartats posteriors.

Altera Monitor Program

Altera ha creat aquest programa per facilitar als estudiant la manipulació de la placa. El podem baixar gratuïtament de la web del fabricant [1]. Amb el seu manual [6], que també el tenim a la web, podem fer els primers projectes seguint els exemples.

Aquesta aplicació ens permet carregar un arxiu en C o en assemblador en el NIOS que prèviament ja tindrem guardat a la DE2. Indicant-li el fitxer .ptf que ens hagi creat el *SOPC Builder* i el fitxer *.c* o *.s* (segons el llenguatge de programació escollit) ens compila i ens carrega el programa principal a la placa. Ens permet fer un debugg utilitzant breakpoint dintre del programa principal.

L'aspecte de la pantalla principal és el següent:

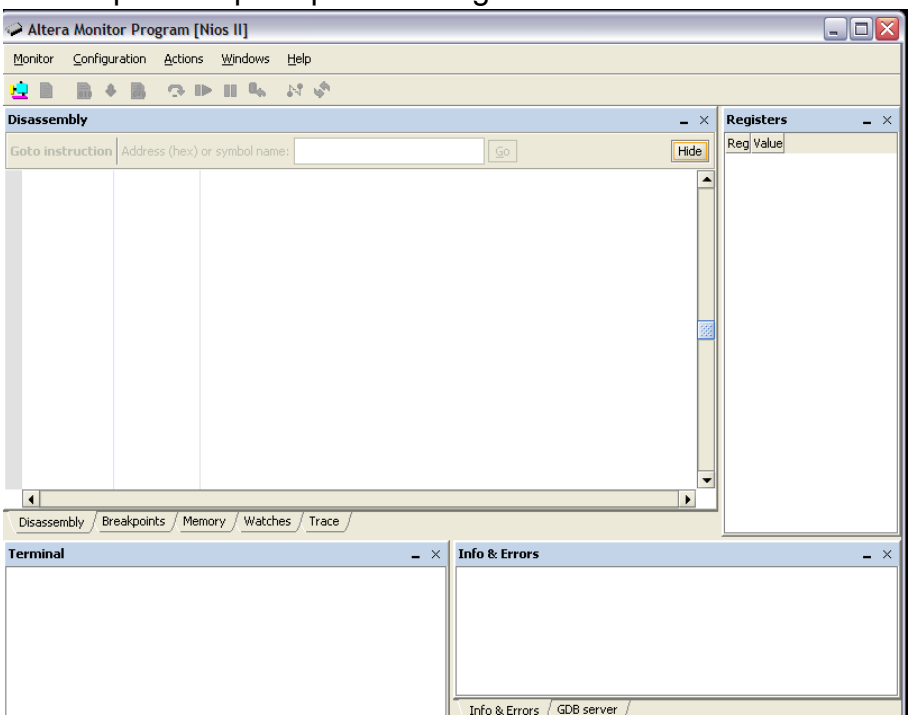


Fig. 1.5. Pantalla principal d'Altera Monitor Program

Veiem a la part superior les icones amb les que podrem carregar, tant l'arxiu .ptf com el que contingui el programa principal. També podrem compilar i carregar amb un *click* a un d'aquests icones. A la part central de la pantalla tindrem la informació del programa carregat i sempre tindrem la possibilitat de que quan aturem la execució ens indiqui en quin punt estem. A la part dreta trobarem informació sobre els valors que prenen els diferents registres implicats en el programa principal. En cas de tindre cap error en la compilació ens apareixerà un missatge a la part inferior dreta de la pantalla principal, ens indicarà el tipus d'error i el nombre de la fila on es troba.

Aquest programa serà el que utilitzarem més endavant amb les nostres aplicacions amb el NIOS.

2. Aplicacions sobre FPGA amb VHDL

Aprendrem a programar diferents aplicacions per la nostra *FPGA*. La placa DE2 d'Altera ens dona la oportunitat de poder provar els nostres programes de manera senzilla i ràpida com veurem a continuació.

2.1. Sistemes combinacionals

En aquest apartat veurem com utilitzar els diferents interruptors per escollir les entrades i les sortides que volem. Veurem con treure el resultat de l'exercici als diferents 7 segments a més d'utilitzar els leds com avisadors. Aprendrem a crear un nou projecte, fer el nostre codi, simular-ho i carregar-ho a la placa.

2.1.1. Declaració I/O, simulació al Quartus II

El que volem veure amb aquest exercici és com podem simular amb el mateix programa sense necessitat de tindre la placa connectada al nostre PC. L'exercici consisteix en assignar cada un dels *LEDs* vermells de la placa a un dels interruptors.

El diagrama de blocs és tant senzill com fer l'assignació directe entre entrades i sortides.

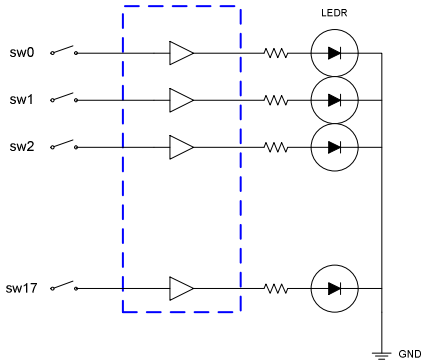


Fig. 2.1. Diagrama de blocs del exercici 1.

Les instruccions que necessitem dintre de la nostra *architecture* seran del tipus:

LEDR(17) <= SW(17)
LEDR(16) <= SW(16)

...
LEDR(0) <= SW(0);

Creació d'un nou projecte al Quartus II

Per començar tenim que tindre una carpeta especifica on guardar tots els arxius que tant nosaltres com el propi *Quartus II* generarà per poder veure els resultats a la placa.

En aquest cas he creat una carpeta amb el nom de Lab_1 on indicaré que es guardin tots els meus fitxers.

El programa ens preguntarà quin nom posarem a la nostra *entity* i quin és el dispositiu que volem programar. En el cas de la placa DE2 escollirem la família: *Cyclone II* i el dispositiu EP2C35F672C6 (Fig. 1.1).

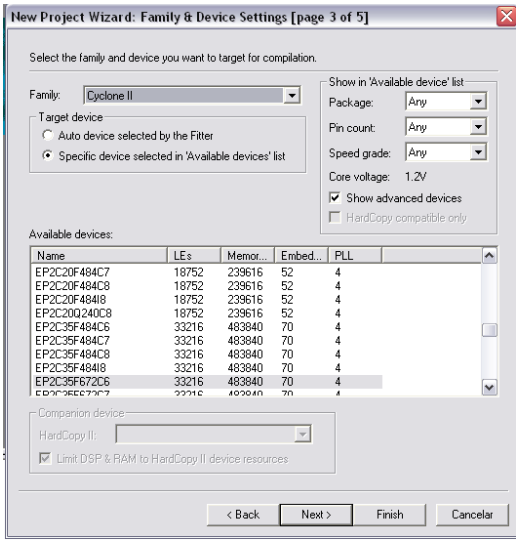


Fig. 2.2. Pantalla on es pot escollir el dispositiu que utilitzarem.

Creació d'una nova entity pel nostre projecte

Creem un nou arxiu .vhd, per fer-ho només tenim que escollir el tipus d'arxiu i guardar-ho amb el nom de l'entitat que em creat per aquest projecte. El codi que s'utilitza és el següent:

```
LIBRARY ieee;
   USE ieee_std_logic_1164.all;
   ENTITY part1 IS
      PORT (   SW : IN STD_LOGIC_VECTOR(17 DOWNTO 0);
               LEDR : OUT STD_LOGIC_VECTOR(17 DOWNTO 0));
   END part1;

ARCHITECTURE Behavior OF part1 IS
```

```
    BEGIN
       LEDR <=SW;
END Behavior
```

Com es pot veure el codi és senzill, només consta d'una entrada i una sortida i una assignació directe entre les dues. Tot i que les entrades siguin un vector de 18 bits no cal fer l'assignació un a un ja que les dues variables tenen el mateix tamany.

Associació entre les variables i els PINS de la placa DE2

Si consultem el manual *Getting started* [12] *,* que trobem al CD que s'incorpora a la caixa o bé a la pàgina web d'Altera [1] podem veure com cada un dels dispositius que hi ha a la placa té una assignació de PIN determinat, per exemple, el SW0 (Interruptor 0) està connectat al PIN N25 i el LEDR0 (Led vermell 0) està connectat al AE23.

Per identificar cada un dels dispositius Altera té una nomenclatura definida, la qual la podem trobar als manuals. Es recomanable utilitzar la mateixa que el fabricant a l'hora de transcriure el nostre codi. El fabricant de la placa ens proporciona un arxiu al CD inclòs amb la placa en el qual s'indiquen tots els pins corresponents a cada un dels dispositius que tenim. Per a que no hi hagin errors a l'hora de fer l'assignació utilitzant aquest arxiu tenim que donar a les variables que utilitzem el nom que el fabricant indica.

Tenim que incorporar l'arxiu al nostre projecte:

Assigment → *import* assignment (Fig. 2.3).

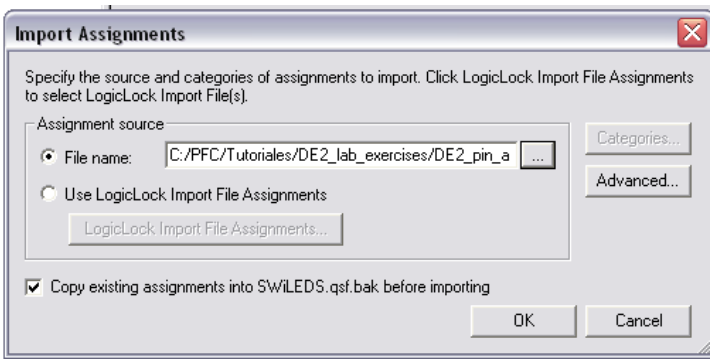


Fig. 2.3.Incorporació de l'arxiu al projecte.

Un cop agregat l'arxiu al projecte es té que tornar a compilar per a que el *Quartus II* ens generi l'arxiu *.sof i .pof* per tal de passar el nostre programa a la placa.

Es normal que quan utilitzem el fitxer que ens dona el fabricant resultin molts '*warnings',* degut a que en aquest apareixen tots els PINS que té la nostra *FPGA*, la gran majoria dels quals no estem utilitzant.

Simulació al Quartus II. Funcinal simulation, timming simulation

Un dels pros del programa *Quartus* és la facilitat per poder comprovar que el disseny de la nostra arquitectura és correcte a través d'una simulació. Tal i com hem vist a la part d'introducció tenim dos tipus de simulació: la *Funtional* i la *Timming.*

Per tal que el simulador ens doni un resultat tenim que crear un arxiu de formes d'ona (Fig. 2.4) per donar valors a les variables d'entrada. Cal indicar les variables que volem veure a la simulació i donar valors a les d'entrada (Fig. 2.5).

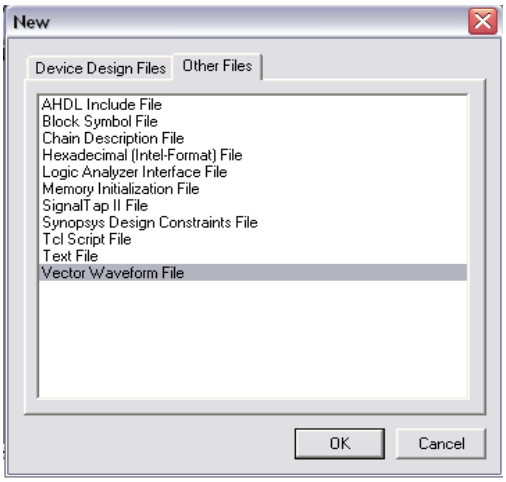


Fig. 2.4. Nou fitxer de Formes d'ona.

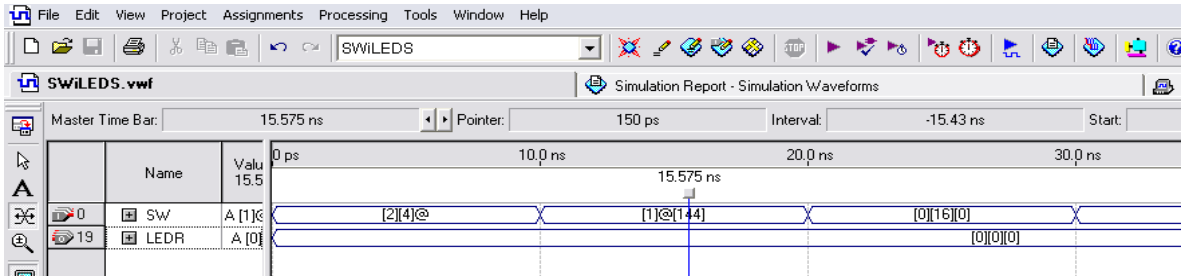


Fig. 2.5. Llistat de variables que veurem a la simulació.

Per tal d'escollir el tipus de simulació, anem a:

Tools → Simulation tool

Tenim que tindre en compte que sempre que escollim la simulació funcional tenim que crear primer un fitxer *Netlist*. Un cop feta la simulació obtenim un

informe de com a anat i les sortides resultants (Fig. 2.6). Veiem que tant les entrades com les sortides són les que esperàvem.

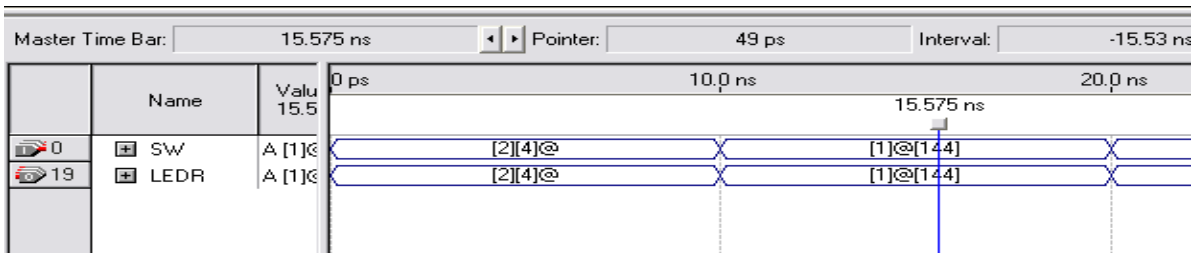


Fig. 2.6. Simulació: Funtional.

En quant a la simulació *Timming* veiem el retard incorporat per les portes lògiques (Fig. 2.7).

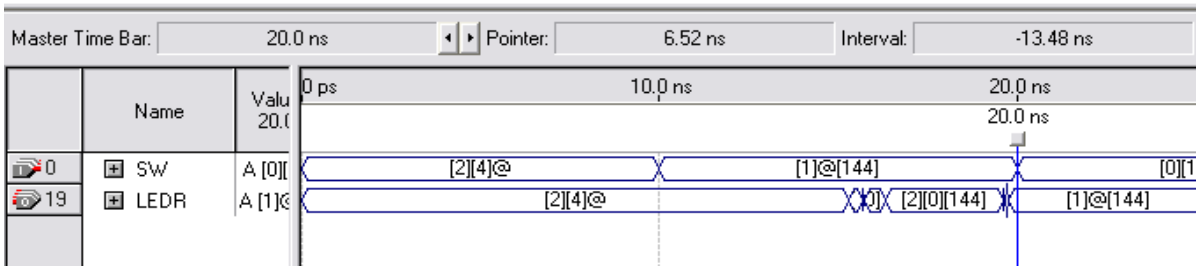


Fig. 2.7. Simulació: Timming

Baixada del nostre projecte a la placa

Per carregar la placa només la tenim que connectar al PC a través del port USB i baixar l'arxiu *.sof* . La placa a de estar connectada a la corrent i l' interruptor a la posició *RUN*.

El resultat ha de ser el que apareix a Fig. 2.8.

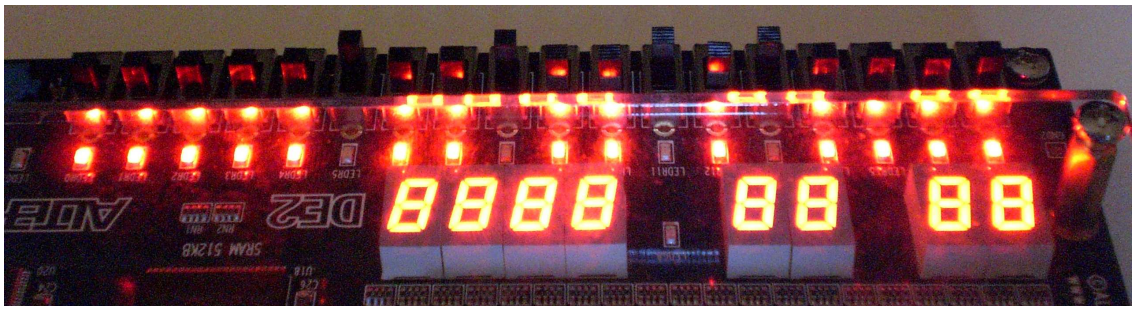


Fig. 2.8. Vista final del projecte a la placa DE2.

2.1.2. Multiplexor de dos entrades. Assignació de PINS

En aquesta part tenim que construir un multiplexor 2-1, amb tres entrades (una de selecció i dos variables a escollir) i una sortida. Veiem les especificacions als esquemes que apareixen a la Fig. 2.9

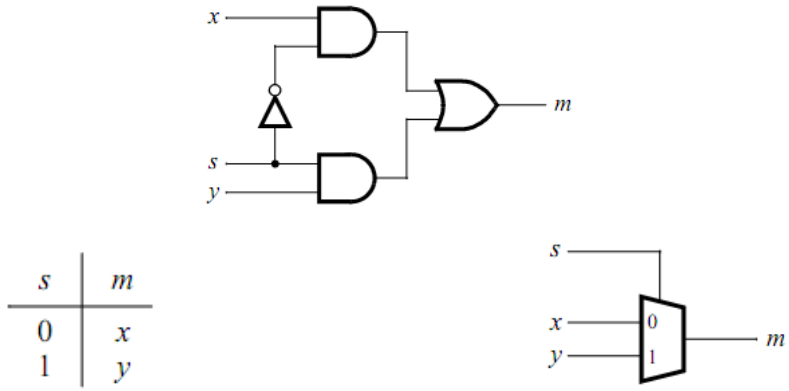


Fig. 2.9. Multiplexor 2-1.

Amb la taula de veritat podem veure com implementar el MUX amb una sola sentència:

```
m <= (NOT (s) AND x) OR (s AND y);
```

La finalitat d'aquest apartat serà aprendre a fer l'assignació de PINS de manera manual, sense l'arxiu que ens proporciona el fabricant.

Creació d'un nou projecte

En aquest cas creem un nou projecte amb el nom de *MUX*, així la nova *entity* també es dirà de la mateixa manera.

Com en el cas anterior és recomanable crear una carpeta amb el nom del projecte on es guardaran tots els arxius que nosaltres mateixos creem i creï el propi *Quartus II.*

Creació de l'arxiu VHDL

Es demana que el SW 17 sigui l'entrada de selecció (s), del SW 7 al 0 la X i del 15 al 8 la Y. Aquestes entrades tenen que estar relacionades amb els LEDR corresponents i la sortida quedarà marcada pels LEDG del 7 al 0.

En aquest cas el codi no es tracta de fer una assignació directe, per facilitar l'escriptura d'aquest, primer, fem un diagrama de flux (Fig. 2.10).

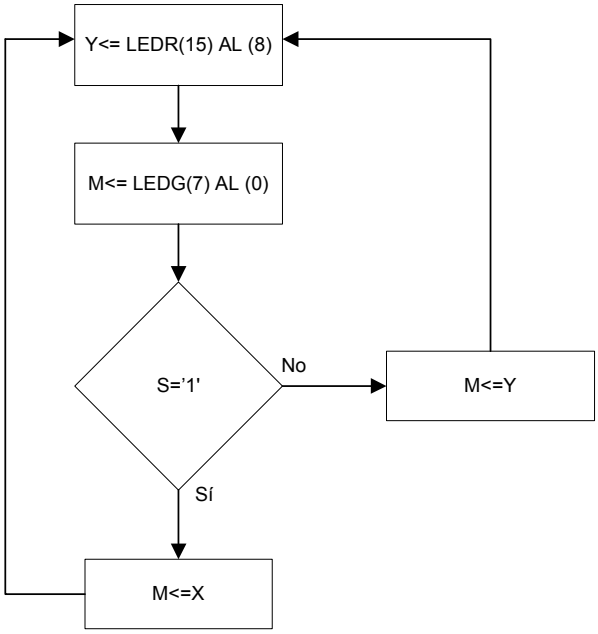


Fig. 2.10. Diagrama de flux del MUX 2-1.

Seguint el diagrama de flux traiem el següent codi:

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY MUX IS
      Port (S    : IN  STD_LOGIC;
        X  : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
        Y  : IN  STD_LOGIC_VECTOR (7 DOWNTO 0);
        LEDR  : OUT    STD_LOGIC_VECTOR (15 DOWNTO 0);
        --LEDG   : OUT    STD_LOGIC_VECTOR (7 DOWNTO 0);
        M  : OUT    STD_LOGIC_VECTOR (7 DOWNTO 0)
        );
END MUX;

ARCHITECTURE MUX_ARCH OF MUX IS
BEGIN
   PROCESS (S)
   BEGIN
      LEDR (15 DOWNTO 8)<= Y;
      LEDR (7 DOWNTO 0)<= X;
      IF (S='1') THEN
         M <= Y;
      ELSE
         M <= X;
      END IF;
   END PROCESS;
END MUX_ARCH;
```

Assignació del PINS

Si hem seguit les indicacions que ens posaven a l'esquema de la Fig. 2.10, les variables no seran les que el fabricant utilitza en el seu arxiu d'assignació de PINS. Per aquesta raó tenim que indicar a la placa manualment aquesta relació. Ho fem de la següent manera:

Consultem el manual *d'Altera* [12] que trobem dins del CD que ens ve amb la placa o bé el baixem de la seva pàgina web [1]. Allà trobem la relació de cada un dels dispositius amb els seus PINS com veiem a la Fig. 2.11.

Signal Name	FPGA Pin No.
SW[0]	PIN_N25
SW[1]	PIN_N26
SW[2]	PIN_P25
SW[3]	PIN_AE14
SW[4]	PIN_AF14
SW[5]	PIN_AD13
SW[6]	PIN_AC13
SW[7]	PIN_C13
SW[8]	PIN_B13
SW[9]	PIN_A13
SW[10]	PIN_N1
SW[11]	PIN_P1
SW[12]	PIN_P2
SW[13]	PIN_T7
SW[14]	PIN_U3
SW[15]	PIN_U4
SW[16]	PIN_V1
SW[17]	PIN_V2

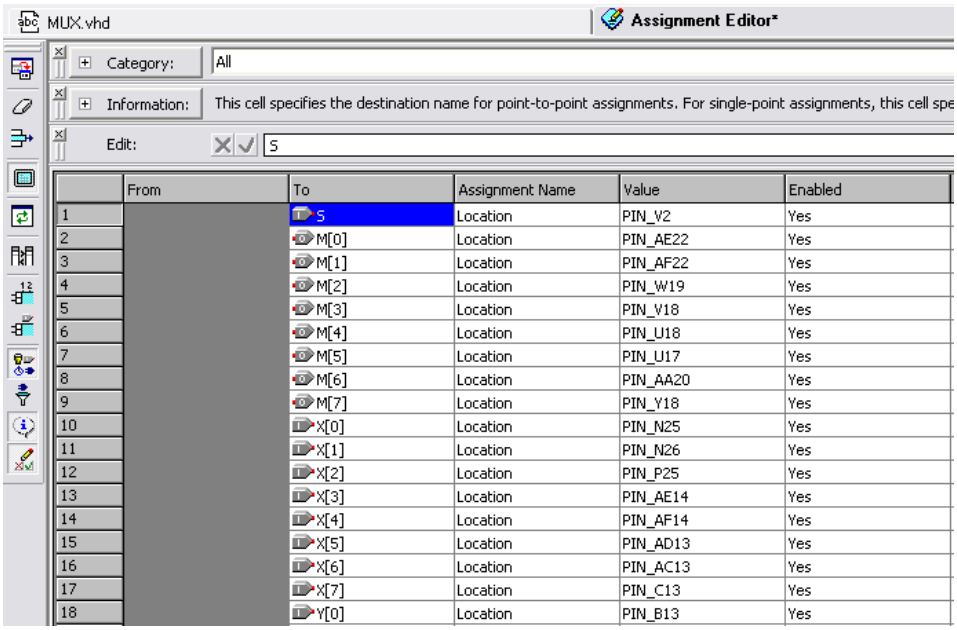
Signal Name	FPGA Pin No.
LEDR[0]	PIN_AE23
LEDR[1]	PIN_AF23
LEDR[2]	PIN_AB21
LEDR[3]	PIN_AC22
LEDR[4]	PIN_AD22
LEDR[5]	PIN_AD23
LEDR[6]	PIN_AD21
LEDR[7]	PIN_AC21
LEDR[8]	PIN_AA14
LEDR[9]	PIN_Y13
LEDR[10]	PIN_AA13
LEDR[11]	PIN_AC14
LEDR[12]	PIN_AD15
LEDR[13]	PIN_AE15
LEDR[14]	PIN_AF13
LEDR[15]	PIN_AE13
LEDR[16]	PIN_AE12
LEDR[17]	PIN_AD12

Fig. 2.11. Taules relació dispositiu- PIN a la placa DE 2.

Per fer la relació manualment el Quartus ens dona una ferramenta que trobem a :

Assigment → Assigment editor.

S'obrirà una finestra com la de la Fig. 2.12 on tenim que omplir el primer i segon camp (*To*, *Location*). En el primer posem la variable que volem assignar i en el segon el PIN de la placa corresponent. Per exemple: la nostra variable de selecció serà la S, en el camp de *To* busquem la S i al camp de *Location* busquem el Pin corresponent al SW 17 PIN_V2. Aquesta operació es té que realitzar amb totes les variables del projecte.



	From	To	Assignment Name	Value	Enabled
1		S	Location	PIN_V2	Yes
2		M[0]	Location	PIN_AE22	Yes
3		M[1]	Location	PIN_AF22	Yes
4		M[2]	Location	PIN_W19	Yes
5		M[3]	Location	PIN_V18	Yes
6		M[4]	Location	PIN_U18	Yes
7		M[5]	Location	PIN_U17	Yes
8		M[6]	Location	PIN_AA20	Yes
9		M[7]	Location	PIN_Y18	Yes
10		X[0]	Location	PIN_N25	Yes
11		X[1]	Location	PIN_N26	Yes
12		X[2]	Location	PIN_P25	Yes
13		X[3]	Location	PIN_AE14	Yes
14		X[4]	Location	PIN_AF14	Yes
15		X[5]	Location	PIN_AD13	Yes
16		X[6]	Location	PIN_AC13	Yes
17		X[7]	Location	PIN_C13	Yes
18		Y[0]	Location	PIN_B13	Yes

Fig. 2.12. Pin Assigment.

Un cop feta aqueta assignació tenim que guardar l'arxiu i compilar de nou el projecte. D'aquesta manera ens crearà l'arxiu *.sof* per carregar-lo a la placa i veure com funciona el nostre projecte. Es recomanable fer una petita simulació abans de carregar el projecte a la placa, tot i que en aquest cas al ser un projecte petit és més còmode comprovar el funcionament directament a la DE2.

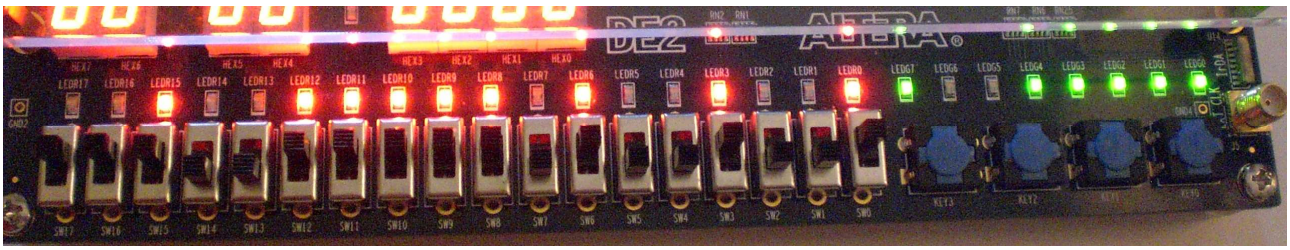
Comprovem el funcionament del projecte

Per tal de carregar la placa, si tenim un altre projecte ja carregat només tenim que apagar i tornar a encendre de nou la DE2 per deixar lliure la memòria, això és possible ja que utilitzem la placa en mode RUN com s'explica en el primer apartat.

Carreguem la placa:

Tools → Programmer

Marquem el *Program/ Configure* i *Start* i comprovem a la placa que tot està correcte.



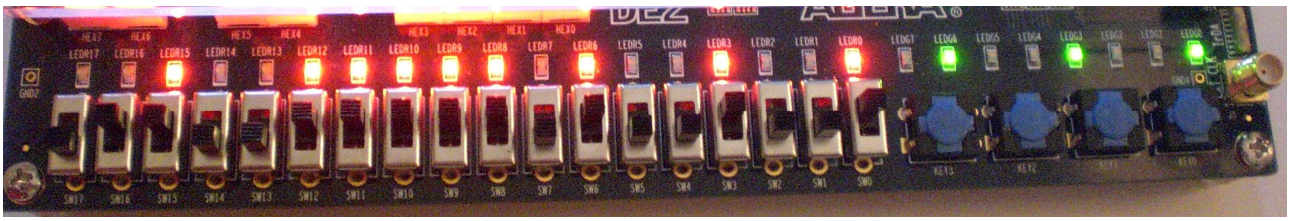


Fig. 2.13. Vista final del projecte 2.

2.1.3. Multiplexor de 5 entrades

En aquesta part es tracta de fer una ampliació de l'apartat anterior, en aquest cas farem un MUX 5-1 amb entrades de 3 bits. Els esquemes de funcionament els tenim a la Fig. 2.14

s_2	s_1	s_0	m
0	0	0	u
0	0	1	v
0	1	0	w
0	1	1	x
1	0	0	y
1	0	1	y
1	1	0	y
1	1	1	y

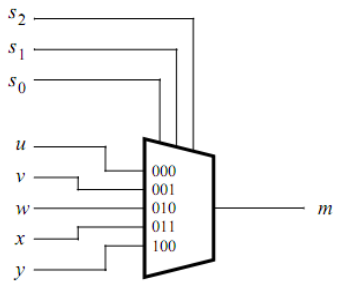


Fig. 2.14. Taula de veritat i simbol del projecte.

Creem el codi VHDL del nostre MUX 5-1

Creem una carpeta i un nou projecte com en els apartats anteriors. En aquets cas ja tenim més variables i és més necessari fer el diagrama de flux (Fig. 2.15) per tindre una guia a l'hora d'escriure el codi.

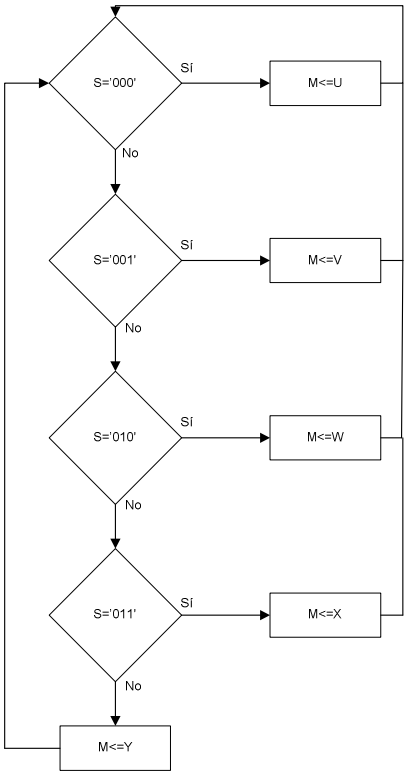


Fig. 2.15. Diagrama de flux del MUX 5-1.

La solució del nou procés utilitzat és la següent:

```
PROCESS (S)
BEGIN
   CASE S IS
      WHEN "000" => M<=U;
      WHEN "001" => M<=V;
      WHEN "010" => M<=W;
      WHEN "011" => M<=X;
      WHEN OTHERS   => M<=Y;
   END CASE;
END PROCESS
```

Guardem l'arxiu VHDL i compilem per comprovar que tot és correcte.

Assignació dels PINS

Com en el cas anterior tenim que fer l'assignació dels Pins manualment. Ho fem de la següent manera: fem correspondre l'entrada de selecció, *S*, amb el SW 17 al 15 i la resta d'entrades (de la *U* a la *Y*) amb els *SW 14* al *0* respectivament. Els *LEDs* vermells els fem correspondre amb els respectius SW i la sortida la representem amb els *LEDs* verds. Finalment el que ens queda és una taula com la que veiem a la Fig. 2.16.

	To	Location	I/O Bank	I/O Standard	General Function	Special Function	Reserved	Enabled
7	W[0]	PIN_AC13	8	3.3-V LVTTL	Dedicated Clock	CLK15, LVDSCLK7p, I...		Yes
8	W[1]	PIN_C13	3	3.3-V LVTTL	Dedicated Clock	CLK10, LVDSCLK5n, I...		Yes
9	W[2]	PIN_B13	4	3.3-V LVTTL	Dedicated Clock	CLK8, LVDSCLK4n, In...		Yes
10	X[0]	PIN_A13	4	3.3-V LVTTL	Dedicated Clock	CLK9, LVDSCLK4p, In...		Yes
11	X[1]	PIN_N1	2	3.3-V LVTTL	Dedicated Clock	CLK1, LVDSCLK0n, In...		Yes
12	X[2]	PIN_P1	1	3.3-V LVTTL	Dedicated Clock	CLK3, LVDSCLK1n, In...		Yes
13	Y[0]	PIN_P2	1	3.3-V LVTTL	Dedicated Clock	CLK2, LVDSCLK1p, In...		Yes
14	Y[1]	PIN_T7	1	3.3-V LVTTL	Row I/O	LVDS15p		Yes
15	Y[2]	PIN_U3	1	3.3-V LVTTL	Row I/O	LVDS17p		Yes
16	S[0]	PIN_U4	1	3.3-V LVTTL	Row I/O	LVDS17n		Yes
17	S[1]	PIN_V1	1	3.3-V LVTTL	Row I/O	LVDS16p		Yes
18	S[2]	PIN_V2	1	3.3-V LVTTL	Row I/O	LVDS16n		Yes
19	LEDR[0]	PIN_AE23	7	3.3-V LVTTL	Column I/O	LVDS151n		Yes
20	LEDR[1]	PIN_AF23	7	3.3-V LVTTL	Column I/O	LVDS151p		Yes
21	LEDR[2]	PIN_AB21	7	3.3-V LVTTL	Column I/O	LVDS152n		Yes
22	LEDR[3]	PIN_AC22	7	3.3-V LVTTL	Column I/O	LVDS152p		Yes
23	LEDR[4]	PIN_AD22	7	3.3-V LVTTL	Column I/O	LVDS153n		Yes
24	LEDR[5]	PIN_AD23	7	3.3-V LVTTL	Column I/O	LVDS153p		Yes
25	LEDR[6]	PIN_AD21	7	3.3-V LVTTL	Column I/O	LVDS154n		Yes
26	LEDR[7]	PIN_AC21	7	3.3-V LVTTL	Column I/O	LVDS154p, CDPCLK3/...		Yes
27	LEDR[8]	PIN_AA14	7	3.3-V LVTTL	Column I/O	LVDS174p		Yes
28	LEDR[9]	PIN_Y13	7	3.3-V LVTTL	Column I/O	LVDS175n		Yes
29	LEDR[10]	PIN_AA13	7	3.3-V LVTTL	Column I/O	LVDS175p		Yes
30	LEDR[11]	PIN_AC14	7	3.3-V LVTTL	Column I/O			Yes
31	LEDR[12]	PIN_AD15	7	3.3-V LVTTL	Column I/O	LVDS176n		Yes
32	LEDR[13]	PIN_AE15	7	3.3-V LVTTL	Column I/O	LVDS176p, DPCLK4/D...		Yes
33	LEDR[14]	PIN_AF13	8	3.3-V LVTTL	Column I/O	LVDS177n		Yes
34	LEDR[15]	PIN_AE13	8	3.3-V LVTTL	Column I/O	LVDS177p, DPCLK3/D...		Yes
35	LEDR[16]	PIN_AE12	8	3.3-V LVTTL	Column I/O	LVDS178n		Yes
36	LEDR[17]	PIN_AD12	8	3.3-V LVTTL	Column I/O	LVDS178p		Yes
37	M[0]	PIN_AE22	7	3.3-V LVTTL	Column I/O	LVDS155n		Yes
38	M[1]	PIN_AF22	7	3.3-V LVTTL	Column I/O	LVDS155p		Yes
39	M[2]	PIN_W19	7	3.3-V LVTTL	Column I/O	LVDS156n		Yes

Fig. 2.16. Assignació de Pins pel MUX 5-1.

Càrrega del projecte a la placa

Com en els casos anteriors carreguem el projecte a la placa i comprovem que el funcionament és correcte. En aquest cas és més probable que ens apareguin errors en l'assignació de PINS ja que tenim més variables i per tant més probabilitat de poder equivocar-nos a l'hora de fer l'assignació manual. Una de les avantatges de la DE2 és que podem tornar a carregar el projecte a la placa sempre que fem una modificació i compilem de nou. *Quartus II* ens modifica el *.sof* i aquets pot substituir al que tenim actualment a la placa.

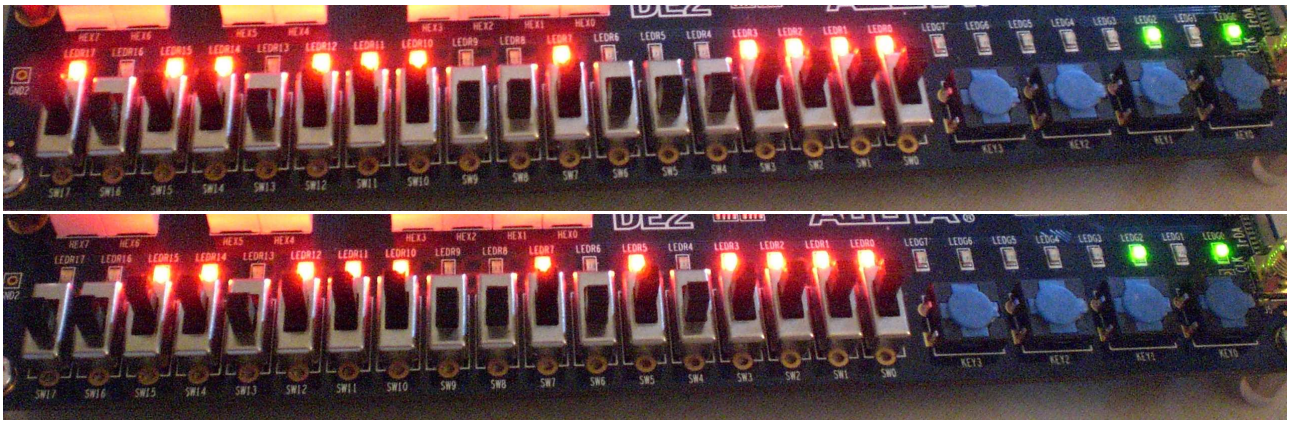


Fig. 2.17. Vista final del projecte MUX 5-1.

2.1.4. Descodificador 7 segments

En aquets apartat estudiarem el funcionament d'un altre dispositiu incorporat a la placa DE2, els Displays 7 segments. En aquest cas utilitzarem un únic Display on vagin apareixent diferents lletres segons la selecció feta amb una determinada entrada. Veiem la taula de veritat a la Fig. 2.18.

$c_2c_1c_0$	Character
000	H
001	E
010	L
011	O
100	
101	
110	
111	

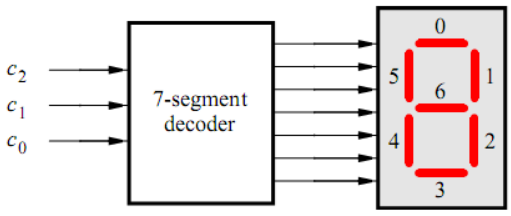


Fig. 2.18. Diagrama de blocs i taula de la veritat.

Creació de Descodificador 7 segments

Volem una entrada de selecció de 3 bits (C) que correspongui amb els SW 2 al 0 i la sortida té que representar-se al HEX0.

En aquest cas és recomanable, ja que és més fàcil, utilitzar les variables tal com el fabricant les nomena en l'arxiu de l'assignació de Pins. Així doncs les entrades i sortides seran:

```
SW: IN STD_LOGIC_VECTOR (2 DOWNTO 0);
HEX0 : OUT STD LOGIC VECTOR(0 TO 6);
```

Per facilitar la creació del codi ens fem una taula de veritat amb el nom de les variables que utilitzem en el nostre projecte:

SW(2)	SW(1)	SW(0)	HEX0(6)	HEX0(5)	HEX0(4)	HEX0(3)	HEX0(2)	HEX0(1)	HEX0(0)
0	0	0	0	0	0	1	0	0	1
0	0	1	1	1	1	1	0	0	1
0	1	0	1	0	0	0	1	1	1
0	1	1	0	1	1	1	1	1	1
1	0	0	1	1	1	1	1	1	1
1	0	1	1	1	1	1	1	1	1
1	1	0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1	1	1

Fig. 2.19. Taula de veritat amb les I/O que utilitzarem al nostre codi.

En aquest projecte es recomanable utilitzar sentencies booleanes (*if, else*, etc), també es té que tindre en compte que els segments estaran encesos si li donem el valor 0 . La solució a la arquitectura és:

```
PROCESS (SW)
  BEGIN
     IF (SW = "000") THEN
        HEX0 <= "0001001";
     ELSE
        IF (SW ="001") THEN
           HEX0 <= "0000110";
        ELSE
           IF (SW ="010") THEN
              HEX0 <= "1000111";
           ELSE
              IF (SW ="011") THEN
                 HEX0 <= "1000000";
              ELSE
                 HEX0 <= "1111111";
              END IF;
           END IF;
        END IF;
     END IF;
END PROCESS;
```

Assignació de Pins i càrrega a la placa

L'assignació de pins la fem utilitzant l'arxiu del fabricant com em fet en apartats anteriors, per això em fet servir els noms adients de les variables a l'hora de crear el codi. Després d'agregar aquest arxiu al nostre projecte tornem a compilar.

Ja tenim l'arxiu .sof apunt per carregar-ho a la placa. Ho fem i veiem com van apareixent les lletres correctament a al nostre Display (Fig. 2.20).

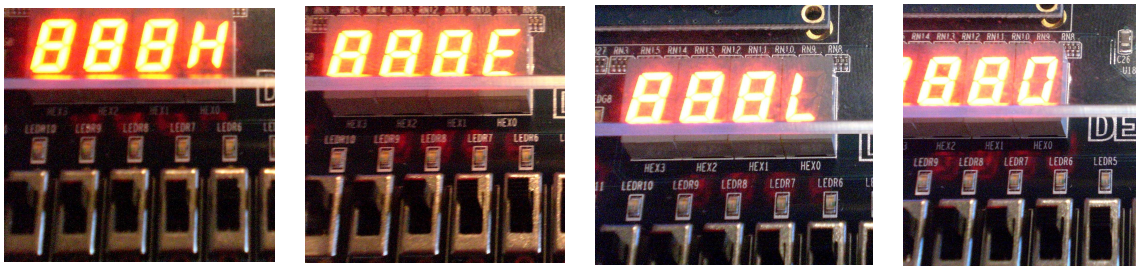


Fig. 2.20. Vista final del projecte. Descodificador 7 segments.

2.1.5. Combinació de dos projectes: MUX + descodificador 7 segments

En aquest cas utilitzarem el que em aprés fins ara i ho integrarem tot en un sol projecte. Utilitzarem el MUX5-1 per fer la selecció de la lletra que volem veure al nostre display tal com apareix a la Fig. 2.21

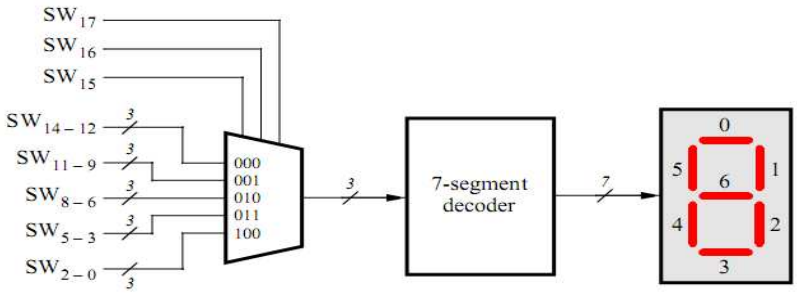


Fig. 2.21. Diagrama de blocs del nostre projecte.

La finalitat és poder construir, amb aquesta combinació la paraula HELLO, de manera que segons la selecció la paraula vagi rotant pels diferents displays. Veiem la taula de veritat a la Fig. 2.22.

SW_{17} SW_{16} SW_{15}	Character pattern				
000	H	E	L	L	O
001	E	L	L	O	H
010	L	L	O	H	E
011	L	O	H	E	L
100	O	H	E	L	L

Fig. 2.22. Taula de veritat del projecte.

Això serà possible fent una combinació amb els arxius que ja em creat als apartats 3 i 4 i amb una nova sentencia: *COMPONENT* la qual ens deixa crear nous components iguals a les *Entitys* creades a altres arxius que tenim agregats al nostre projecte. És molt comú que al integrar varis projectes tinguem que crear noves variables les quals no són ni entrades ni sortides del sistema global, sinó variables internes, *signals*.

Creació d'un nou projecte

Creem una nova carpeta. En aquesta acabarem adjuntant els arxius VHDL de l'apartat 3 i 4 on em creat el MUX 5-1 i el descodificador 7 Segments:

Project → Add/ Remove files to project

Creació d'un nou arxiu VHDL

En aquest apartat tenim que crear nous components (nous descodificadors i nous multiplexors). És recomanable fer, en primer lloc, un projecte en el que només es creï un *MUX* i un descodificador per veure el funcionament de la sentencia *Component, Port Map* i la creació de *Signals*.

```
ARCHITECTURE Behavior OF part5 IS
COMPONENT mux 3bit 5to1
   PORT(S,U,V,W,X,Y :IN STD LOGIC VECTOR(2 DOWNTO 0);
   M : OUT STD LOGIC VECTOR(2 DOWNTO 0));
   END COMPONENT;
   COMPONENT char 7seg
   PORT ( C : IN STD LOGIC VECTOR(2 DOWNTO 0);
   Display : OUT STD LOGIC VECTOR(0 TO 6));
END COMPONENT;

SIGNAL M : STD LOGIC VECTOR(2 DOWNTO 0);
   BEGIN
   M0: mux 3bit 5to1 PORT MAP (SW(17 DOWNTO 15), SW(14
DOWNTO 12), SW(11 DOWNTO 9),
   SW(8 DOWNTO 6), SW(5 DOWNTO 3), SW(2 DOWNTO 0), M);
   H0: char 7seg PORT MAP (M, HEX0);
END Behavior;
```

Un cop tinguem clar el funcionament d'aquestes noves sentencies ens fem un esquema per veure com podem fabricar el nostre projecte i determinar les entrades, sortides i senyals dels sistema (Fig. 2.23).

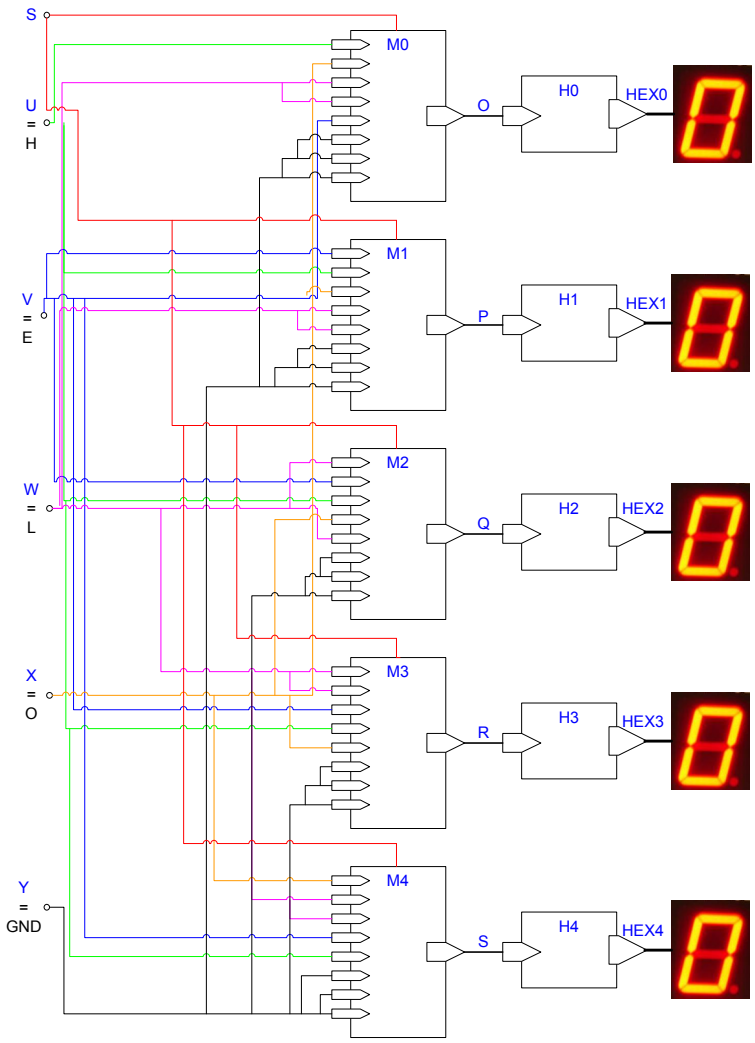


Fig. 2.23. Diagrama de blocs del projecte final.

Com es veu a l'esquema determinem que necessitem 5 senyals que enllaçaran els MUX amb els descodificadors i que les entrades a cada MUX variaran segons l'entrada de selecció i la sortida que es vulgui.

La solució pel codi no és més que una ampliació de l'exemple anterior amb un sol component. En aquest cas posem un exemple de dos dels 5 7-segments que utilitzarem:

```
      M0: MUX5 PORT MAP (SW(17 DOWNTO 15), SW(14 DOWNTO 12), SW(5
DOWNTO 3),
      SW(8 DOWNTO 6), SW(8 DOWNTO 6), SW(2 DOWNTO 0), O);
      H0: SEVSEG PORT MAP (O, HEX4);

      M1: MUX5 PORT MAP (SW(17 DOWNTO 15) ,SW(11 DOWNTO 9), SW(14
DOWNTO 12),
      SW(5 DOWNTO 3), SW(8 DOWNTO 6), SW(8 DOWNTO 6), P);
      H1: SEVSEG PORT MAP (P, HEX3);
```

Assignació de pins i càrrega del projecte a la placa

Adjuntem al projecte el *Pin_assigment* del fabricant i compilem com hem fet fins ara. Carreguem de la mateixa manera l'arxiu *.sof* a la placa i comprovem que fent diferents seleccions veiem als 7 segments la paraula *HELLO* (Fig. 2.24).

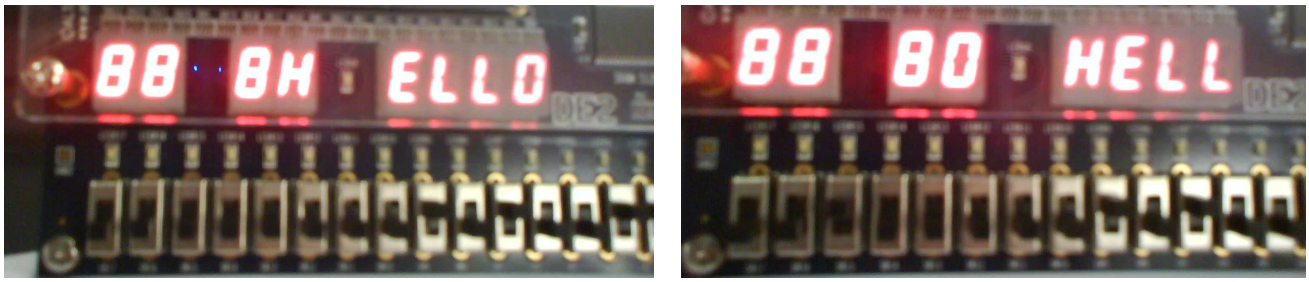


Fig. 2.24. Vista final del projecte. Rotació de la paraula HELLO.

2.1.6. Utilització de tots els 7 segments de la placa DE2

Es proposa la ampliació del projecte anterior fent servir totes les combinacions possibles de les entrades de selecció i tots els Displays de la Placa DE2. La taula de veritat és la que es veu a la Fig. 2.25

SW_{17} SW_{16} SW_{15}	Character pattern							
000				H	E	L	L	O
001			H	E	L	L	O	
010		H	E	L	L	O		
011	H	E	L	L	O			
100	E	L	L	O				H
101	L	L	O				H	E
110	L	O				H	E	L
111	O				H	E	L	L

Fig. 2.25. Taula de veritat del projecte.

Com en el cas anterior es tracta de fer l'assignació correcte al *Port Map* per tal de que la sortida sigui la que toca quan escollim una entrada de selecció determinada. L'esquema seria igual al de l'apartat anterior ampliant el nombre de *MUX* i descodificadors a 8. A més cal també declarar més senyals per poder transmetre les sortides dels *MU*X a les entrades dels descodificadors.

El funcionament del projecte és el que podem veure a la Fig. 2.26

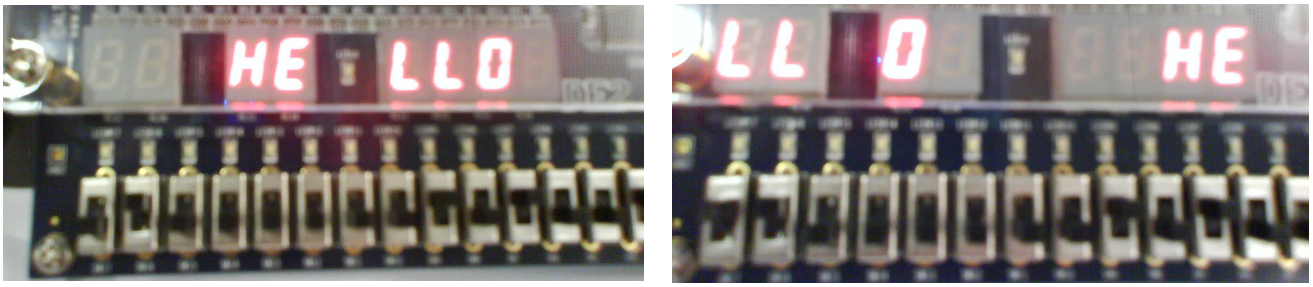


Fig. 2.26. Vista final del projecte. Ampliació de la rotació de la paraula HELLO

2.2. Sistemes seqüencials

En aquest exercici veurem com podem crear un comptador de segons, depenent del *clock* que utilitzem com a base de temps.

La nostra placa Altera DE2 ens dona tres possibilitats pel que es refereix a la base de temps. Conta amb dos *clocks* interns i un d'extern. Els interns ens donen una freqüència de 27 MHz i una de 50 MHz. El tercer depèn de l'entrada que connectem al port *SMA* de la placa.

En el nostre exercici el que volem és fer un comptador de segons (cronòmetre). Ho farem utilitzant com a base de temps la del primer *clock* que ens proporciona la pròpia placa, de tal manera que la freqüència base del *clock* serà de 27 MHz i la tindrem que adaptar al nostre projecte fent un divisor de freqüència.

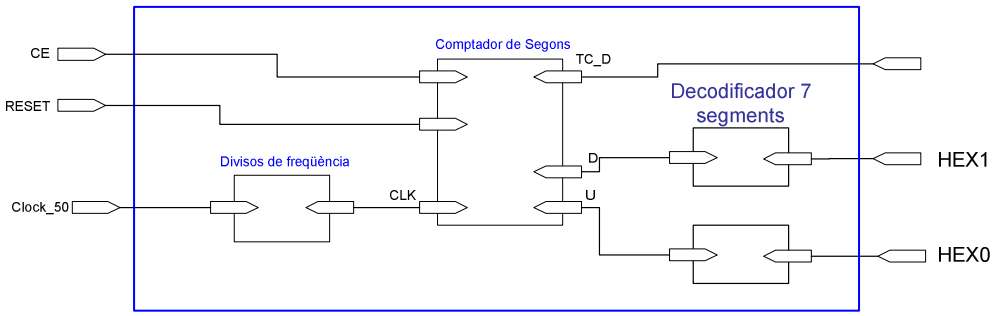


Fig. 2.27. Diagrama de blocs del nostre comptador de segons.

2.2.1. Divisor de freqüència

Divisor de freqüència. Ja que farem un comptador de segons, necessitem que la nostra freqüència de rellotge sigui de 1 Hz per tal de que cada segon s'incrementi el comptador.
El diagrama de blocs d'aquest divisor és molt senzill
(

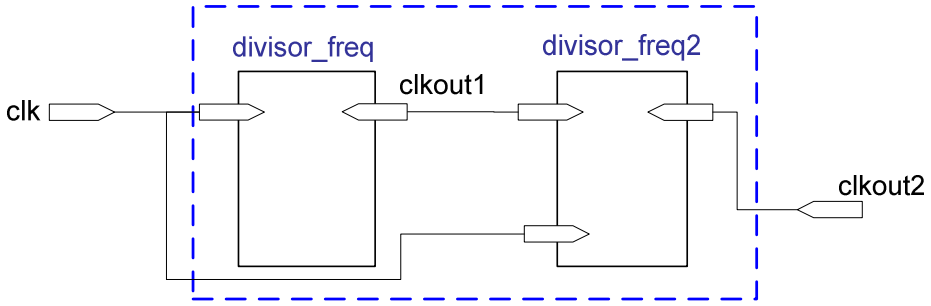


Fig. 2.28)

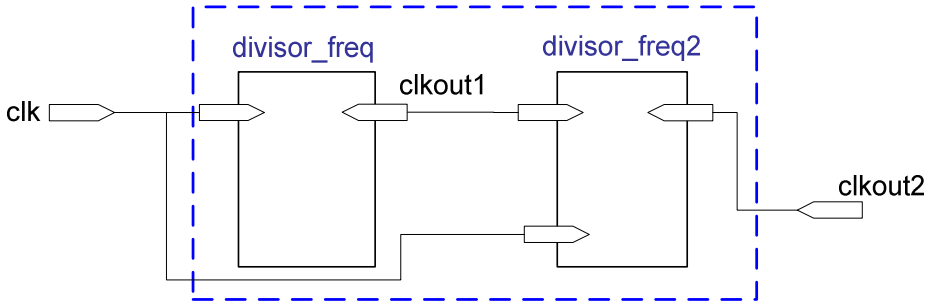


Fig. 2.28. Diagrama de blocs del divisor de freqüència.

Per fer això tenim que tindre en compte per quant volem dividir la nostra freqüència sempre tenint en compte que tenen que ser números parells. A la pàgina web de l'assignatura de SED [2] podem trobar el codi per fer aquest divisor:

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;

ENTITY div_freq IS
   PORT (   clk      : IN STD_LOGIC;
         clkout1,clkout2   : OUT STD_LOGIC);
END div_freq;

ARCHITECTURE funcional OF div_freq IS

CONSTANT COMPTATGE : INTEGER := 2500000;
CONSTANT COMPTATGE2 : INTEGER := 20;

SIGNAL clkdiv : STD_LOGIC; -- senyal per saber quan el comptador
és a la meitat
SIGNAL   clkdiv2   :   STD_LOGIC;   --   senyal   per   saber   quan   el
comptador és a la meitat

SIGNAL divlimit : INTEGER RANGE COMPTATGE DOWNTO 0 := 1;
SIGNAL divlimit2 : INTEGER RANGE COMPTATGE2 DOWNTO 0 := 1

BEGIN
   divisor_freq: PROCESS (clk)
      BEGIN
         IF (clk'EVENT and clk = '1') THEN
            IF divlimit /= COMPTATGE THEN
               divlimit <= divlimit +1;
```

```
            ELSE
               divlimit <= 1;
clkdiv <= NOT clkdiv;
            END IF;
         END IF;
   END PROCESS divisor_freq;

   clkout1 <= clkdiv;
-------------------------------------------------------------------
   divisor_freq2:PROCESS (clkdiv)
      BEGIN
         IF (clkdiv'EVENT and clkdiv = '1') THEN
            IF divlimit2 /= COMPTATGE2 THEN
               divlimit2 <= divlimit2 +1;
            ELSE
               divlimit2 <= 1;
   clkdiv2 <= NOT clkdiv2;
            END IF;
         END IF;
   END PROCESS divisor_freq2;
   clkout2 <= clkdiv2;
-------------------------------------------------------------------
END functional;
```

En el cas d'aquest codi veiem que hi ha dos variables que divideixen la nostra freqüència:

```
CONSTANT COMPTATGE : INTEGER := 2500000;
  CONSTANT COMPTATGE2 : INTEGER := 20;
```

Així si tenim una freqüència de 50MHz /2500000x20 obtenim una freqüència de 1 Hz.

Per tal d'escollir la freqüència que ens dona la pròpia placa de 50MHz, a l'hora de fer l'assignació de pins tenim que escollir l'adient pel nostre *clock* intern (PIN_N2).

Signal Name	FPGA Pin No.	Description
CLOCK_27	PIN_D13	27 MHz clock input
CLOCK_50	PIN_N2	50 MHz clock input
EXT_CLOCK	PIN_P26	External (SMA) clock input

Fig. 2.29. Assignació de pins pels Clocks de la placa DE2.

Un cop tenim el nostre codi inserit en un projecte VHDL el que podem fer per comprovar el funcionament és simular-ho. El *Quartus II,* a l'hora de fer el fitxer de formes d'ona ens dona la possibilitat de crear un *clock*, només tenim que dir quina és la freqüència o, en aquest cas, període (Fig. 2.30)

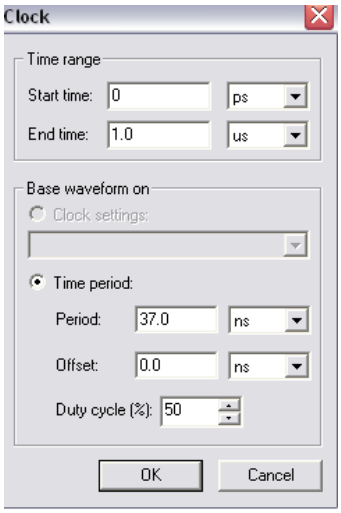


Fig. 2.30. Creació d'un clock de 37ns, 27MHz

La problemàtica de la simulació és que el període de temps que podem veure és massa petit per comprovar el correcte funcionament amb les dades originals. Per aquest motiu el que podem fer per veure si el codi és correcte és modificar el nombre pel que es divideix la freqüència i així comprovar que es fa el que volem tot i no ser la freqüència desitjada. En aquets cas posem que:

```
CONSTANT COMPTATGE : INTEGER := 2;
CONSTANT COMPTATGE2 : INTEGER := 3;
```

En aquets cas si que veiem les oscil·lacions de sortides tenen un període més gran que la d'entrada, així el nostre divisor de freqüència funciona correctament. Podem veure el resultat de la simulació a la Fig. 2.31, on s'aprecia que els *clocks* de sortida són més grans que el d'entrada.

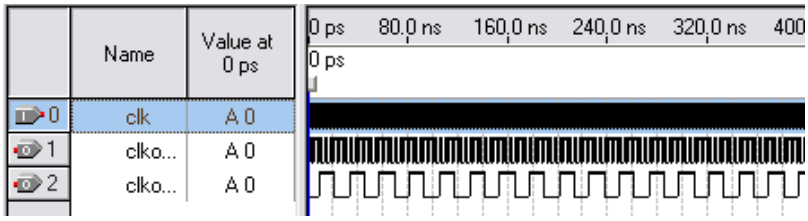


Fig. 2.31. Simulació del divisor de freqüència.

2.2.2. Comptador de 4 bits

Veiem con podem implementar un comptador que ens doni una senyal de sortida cada cop que ens hagi arribat a '9'. El diagrama de blocs serà el següent:

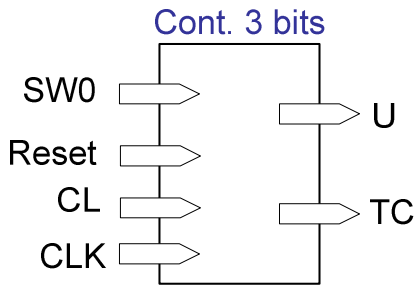


Fig. 2.32. Diagrama de blocs del comptador de 4 bits.

Com podem veure tindrem una entrada de *Reset* per posar a '0' la sortida així com una entrada que ens vindrà directe del interruptor de la placa que posarà en funcionament el comptador. Les sortides del sistema seran dues, una que ens donarà el valor del número, pel que és un cadena de 4 bits i la que es posarà a '1' per indicar-nos que hem arribat a nou.

Creació del projecte i de l'arxiu VHDL corresponent

Per crear aquest codi farem servir una FSM (*Finite State Machine*) que no és més que una màquina d'estats la qual es descriu per mòduls. Aquest diferents mòduls són molt diferenciats i cada un d'ells compleix una funció determinada dintre de la màquina, pel que necessiten que les entrades també es distribueixin segons les feines a realitzar per cada una de les rutines. Podem veure al següent esquema el funcionament d'una Màquina finita per aquesta aplicació en concret:

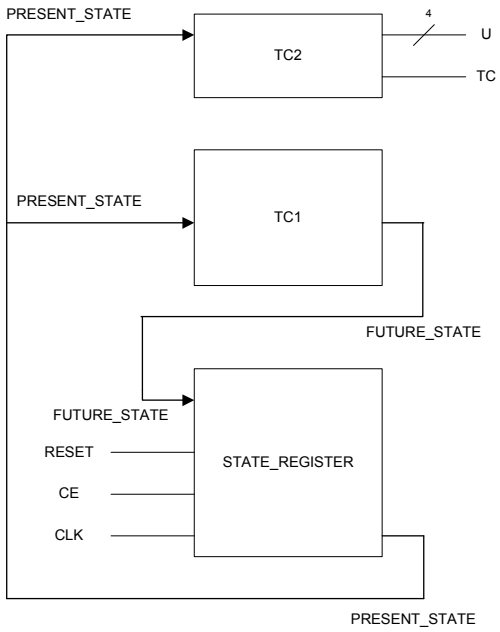


Fig. 2.33. Diagrama de blocs de la FSM.

Vist des del punt de vista d'una Màquina d'estats finits, la construcció del codi és una feina senzilla. Veiem les funcionalitats de cada un dels blocs en

diferents diagrames de flux:

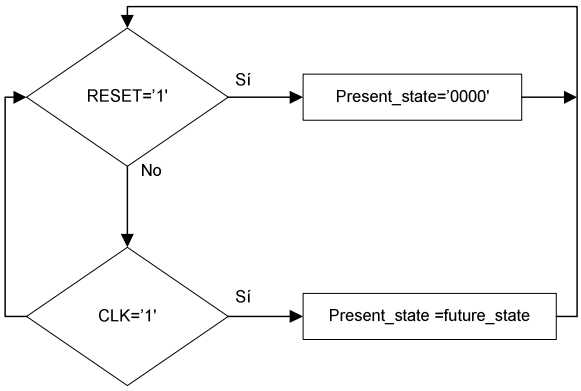


Fig. 2.34. STATE_REGISTER

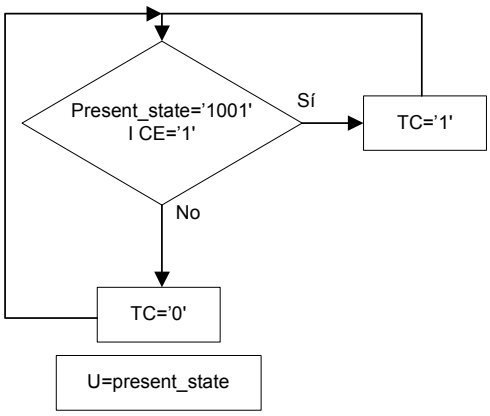


Fig. 2.35. TC2

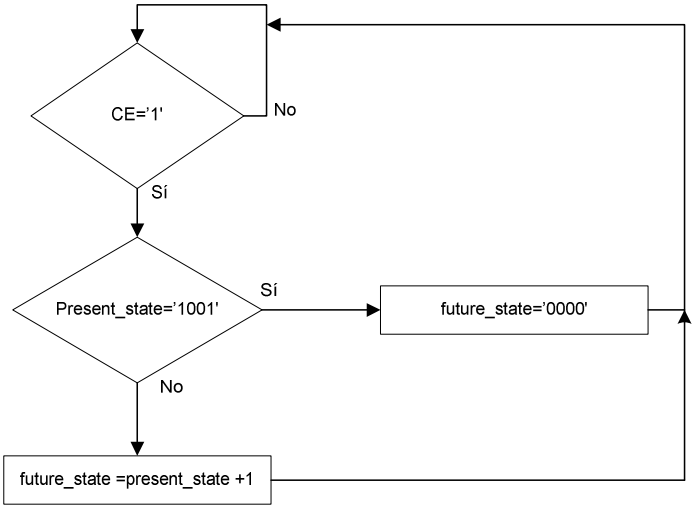


Fig. 2.36. TC1

A partir d'aquí extreure el codi és una feina senzilla. Veiem que tenim diferents senyals que ens proporcionen l'estat en que es troba el comptador i el que es tribarà al següent flanc de pujada del rellotge.
Veiem el codi:

```
LIBRARY IEEE;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY contador IS
      PORT(RESET,CLK,CE   :  IN  std_logic;
           TC    :  OUT std_logic;
           U     :  OUT std_logic_vector(3 DOWNTO 0);
           CL    :  IN STD_LOGIC
          );
END contador;
```

```
ARCHITECTURE FSM_contador OF contador IS

   SIGNAL present_state, future_state: std_logic_vector(3 DOWNTO 0);

BEGIN

-------------------------------------------------------------------
state_register: PROCESS (RESET,CLK)
   BEGIN
      IF (RESET='1' OR CL='1') THEN
         present_state <= "0000";
      ELSIF (CLK='1' AND CLK'event) THEN
         present_state<=future_state;
      END IF;
   END PROCESS state_register;
-------------------------------------------------------------------
TC1: PROCESS (present_state,CE)
   BEGIN
      IF CE = '1' THEN
         IF (present_state = "1001" ) THEN
            future_state <= "0000";
         ELSE
            future_state <= present_state + 1;
         END IF;
      ELSE
         future_state <= present_state;
      END IF;
   END PROCESS TC1;
-------------------------------------------------------------------
---TC2:
---SI CONTINUEM COMPTANT PERÒ I S'ARRIBA A 9 EN TC ENS AVISARÀ
   TC <= '1' WHEN (present_state = "1001" AND CE = '1') ELSE '0';
   U <= present_state;
```

Simulació del projecte

En aquest cas no el provem a la placa degut a que no tenim cap visualitzador que ens ajudi a veure-ho.

Creem un arxiu de formes d'ona i fem la simulació fixant-nos en l'activació del '*TC*' quan '*U*' arriba a '9'. Veiem el resultat de la simulació funcional:

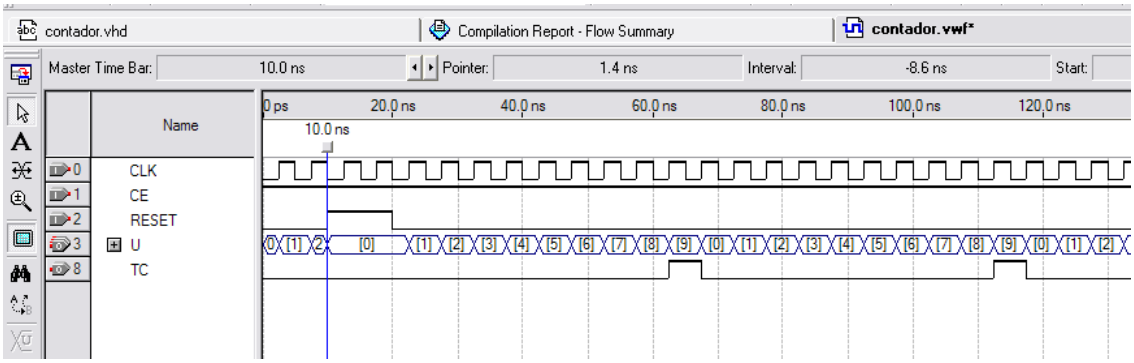


Fig. 2.37. Simulació funcional del comptador

2.2.3. Visualitzador dels segons

Descodificador 7 segments. Volem que els segons comptabilitzats en l'apartat anterior apareguin als nostres 7 Segments. Per això tenim que fer la traducció de binari a 7 segments. Aquest exercici és molt semblant al descodificador que es va fer a l'apartat 2.1.4.

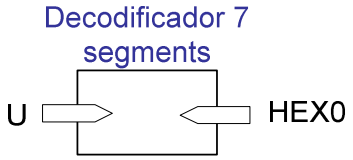


Fig. 2.38. Diagrama de blocs del descodificador 7 segments.

Es té que recordar que els pins del 7 segment de la nostra placa estaran encesos en el cas que el valor d'entrada sigui 0. Per això ens fem una taula amb els valors que tindrem a l'entrada del sistema i els que tindran que ser de sortida (Fig. 2.40):

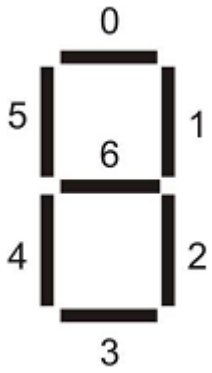


Fig. 2.39. Distribució dels bits del 7-segments

Dígit	HEX0
0	1000000
1	1111001
2	0100100
3	0110000
4	0011001
5	0010010
6	0000010
7	1111000
8	0000000
9	0011000

Fig. 2.40. Taula de veritat del descodificador.

Veiem el codi que hem fet:

```
LIBRARY ieee;
USE  IEEE.STD_LOGIC_1164.all;
USE  IEEE.STD_LOGIC_ARITH.all;
USE  IEEE.STD_LOGIC_UNSIGNED.all;

ENTITY descodificador IS
   PORT (   digit: IN STD_LOGIC_VECTOR (3 DOWNTO 0);
         HEX0: OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
         );
END descodificador;
ARCHITECTURE desc_arch OF descodificador IS
```

```
BEGIN
   PROCESS (digit)
   BEGIN
      IF (digit = "0000") THEN
         HEX0 <= "1000000";

         ELSIF (digit ="0001") THEN
         HEX0 <= "1111001";

         ELSIF (digit ="0010") THEN
         HEX0 <= "0100100";

         ELSIF (digit ="0011") THEN
         HEX0 <= "0110000";

         ELSIF (digit ="0100") THEN
         HEX0 <= "0011001";

         ELSIF (digit ="0101") THEN
         HEX0 <= "0010010";

         ELSIF (digit ="0110") THEN
         HEX0 <= "0000010";

         ELSIF (digit ="0110") THEN
         HEX0 <= "0000010";

         ELSIF (digit ="0111") THEN
         HEX0 <= "1111000";

         ELSIF (digit ="1000") THEN
         HEX0 <= "0000000";

         ELSIF (digit ="1001") THEN
         HEX0 <= "0011000";

      END IF;
   END PROCESS;

END desc_arch;
```

Utilitzem un *process* ja que volem que la sortida variï cada cop que es fa un canvi a la entrada del descodificador i que ens aparegui per pantalla el valor nou.

2.2.4. Comptador de segons

En aquest apartat ens carregarem de crear un comptador de segons, és a dir de ‘00’ a ‘59’. En un dels apartats anteriors (2.2.2) hem creat un comptador de només un dígit, en aquest cas tenim que, aprofitant aquest mòdul del comptador, crear un de dos dígits. Veiem l’estructura bàsica i interna que tindrà el nou projecte:

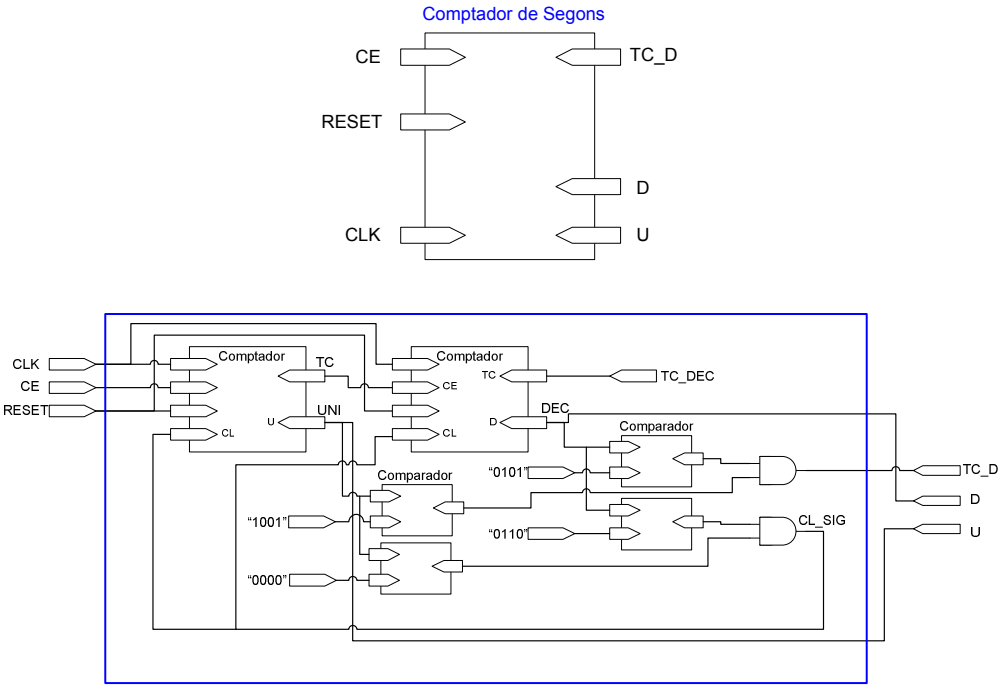


Fig. 2.41. Diagrama de blocs general i intern del comptador de segons.

A partir del diagrama general del sistema podem treure el codi que fa referència a la *entity* d'aquest mòdul. Del segon diagrama veiem les operacions lògiques que tenim que realitzar per a que el bloc funcioni correctament. Tenim comparadors que al nostre codi es traduiran com a simples '='. Tenim senyals que controlaran el funcionament dels dos comptadors interns com ara: CL_SIG que ens posarà el comptador a '00' quan tinguem el valor de '60'. També tenim la senyal TC_D que ens donarà un '1' quan la sortida sigui de '59', veurem que aquesta ens serà útil més endavant quan creem nous mòduls. Veiem el codi que em fet a partir d'aquest esquema:

```
LIBRARY IEEE;
   USE IEEE.STD_LOGIC_1164.ALL;
   USE IEEE.STD_LOGIC_ARITH.ALL;
   USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY contador_segons IS
      PORT( RESET,CLK,CE   :  IN  std_logic;
            TC_D           :  OUT std_logic;
            U, D           :  OUT std_logic_vector(3 DOWNTO 0)
          );
END contador_segons;

ARCHITECTURE FSM_contador_segons OF contador_segons IS

   SIGNAL TC, TC_DEC    : STD_LOGIC;
   SIGNAL UNI, DEC      : STD_LOGIC_VECTOR (3 DOWNTO 0);
   SIGNAL CL_SIG        : STD_LOGIC;
-----------------------------------------------------------------
```

```
   COMPONENT contador
      PORT(
            RESET,CLK,CE   :  IN  std_logic;
            TC             :  OUT std_logic;
            U              :  OUT std_logic_vector(3 DOWNTO 0);
            CL             :  IN STD_LOGIC
            );
   END COMPONENT;

   ----------------------------------------------------------------
BEGIN

   Unidades    :  contador PORT  MAP   (RESET, CLK, CE, TC,  UNI,
CL_SIG);
   Decenes     :  contador PORT  MAP (RESET, CLK, TC, TC_DEC, DEC,
CL_SIG);

   CLR : PROCESS (CLK, UNI, DEC)
   BEGIN
      IF (UNI="0000" AND DEC ="0110") THEN
         CL_SIG <='1';
      ELSE
         CL_SIG <='0';
      END IF;

   END PROCESS;

U <= UNI;
D <= DEC;
TC_D <= '1' WHEN (UNI ="1001" AND DEC = "0101")ELSE '0';

   ----------------------------------------------------------------
END FSM_contador_segons;
```

El motiu de crear una senyals a les que després se li fa una assignació directe es degut a que les variables creades com a sortides el Quartus no les permet llegir. Per poder crear el CL i el *Terminal Count* (TC_D) necessitàvem llegir les sortides dels comptadors. Per aquesta raó hem creat les senyals DEC i UNI.

Per comprovar el funcionament fem una simulació. Comprovem que es genera un pols cada cop que arribem a '59' ja que en posteriors projectes aquesta funció serà molt útil.

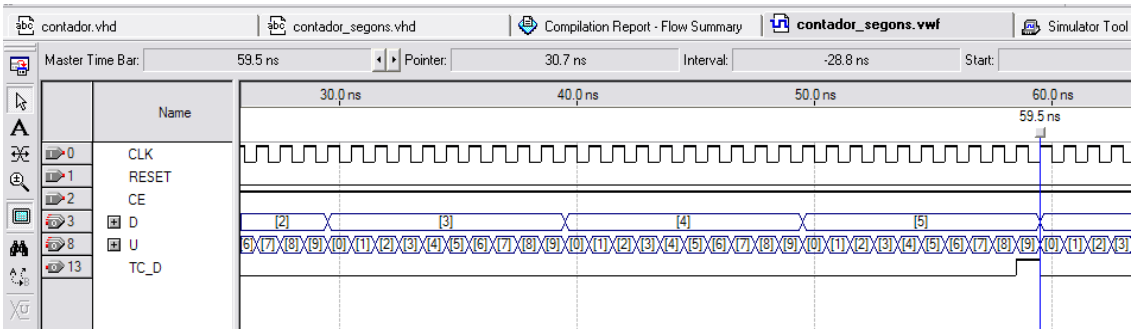


Fig. 2.42. Simulació del comptador de segons.

Visualització a la placa

En aquest apartat utilitzarem el divisor de freqüències i el descodificador 7 segments per poder veure el resultat directament a la placa. Podríem fer la simulació al *Quartus* tot i que el resultat no es podria comprovar de manera immediata, sinó que tindríem que comprovar que les sortides dels 7 segments són els correctes bit a bit.

La placa DE2, és fàcil de carregar, així que aprofitem aquesta facilitat per comprovar-lo directament a la placa.

En aquest nou fitxer *.vhd* que creem per inserir tots dins d'un de sol, indiquem una nova *entity* global del sistema i una determinada assignació de ports com veiem a continuació. Recordem que el sistema és el que tenim al inici d'aquest apartat Fig. 2.27

```
ENTITY COMP_VISUAL IS
   PORT (
         CE, RESET    :  IN std_logic;
         CLOCK_50     :  IN std_logic;
         TC_D         :  OUT std_logic;
         HEX0, HEX1   :  OUT std_logic_vector (6 DOWNTO 0)
      );
END COMP_VISUAL;

   -------------------------------------------------------------------
... declaració de components
   -------------------------------------------------------------------

   freq     :  divfreq     PORT MAP (CLOCK_50,CLK1, CLK2);
   compta   :  contador_segons PORT MAP (RESET,CLK2,CE, TC_D, U, D);
   des_unitats :  descodificador PORT MAP (U, CLK2, HEX0);
   des_desenes :  descodificador PORT MAP (D, CLK2, HEX1);
```

Recordem que abans de carregar el .sof a la placa necessitem fer l'assignació de pins, guardar i compilar de nou per a que la placa reconegui les entrades i sortides. Veiem el resultat:

Fig. 2.43. Visualització final del projecte.

2.3. Rellotge digital

En aquest apartat volem construir, amb els coneixements i dispositius que hem creat fins ara el nostre propi rellotge digital. Al final tindrem un rellotge que podrem posar en hora per mitjà dels botons. Veiem el diagrama de blocs del sistema complert:

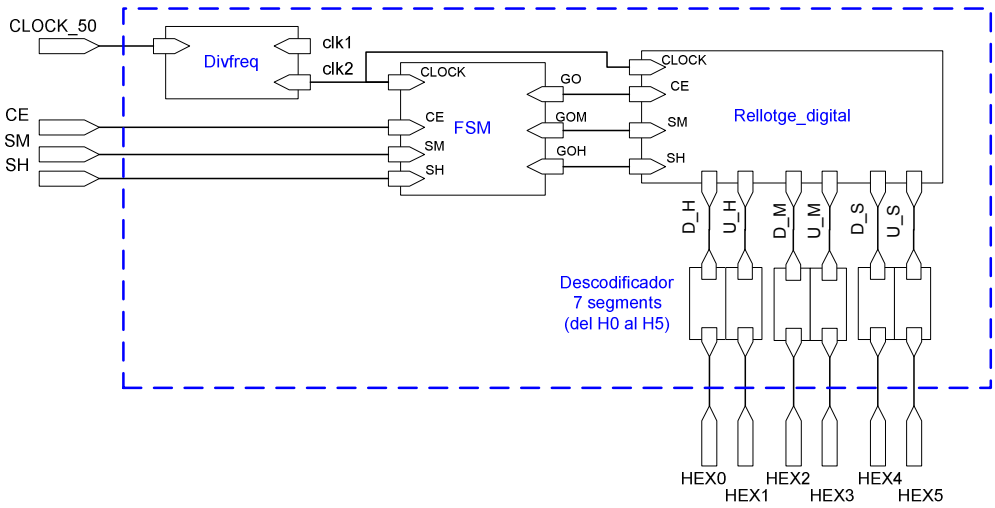


Fig. 2.44. Diagrama de blocs del rellotge digital

2.3.1. Comptador de hores

Per aquest comptador els canvis que tenim són mínims ja que als apartats anterior hem vist com crear el comptador fins a '59', ara només ens cal modificar el codi per a que torni a '00' al valor final de '23'.

Veiem la simulació de la part del comptador d'hores:

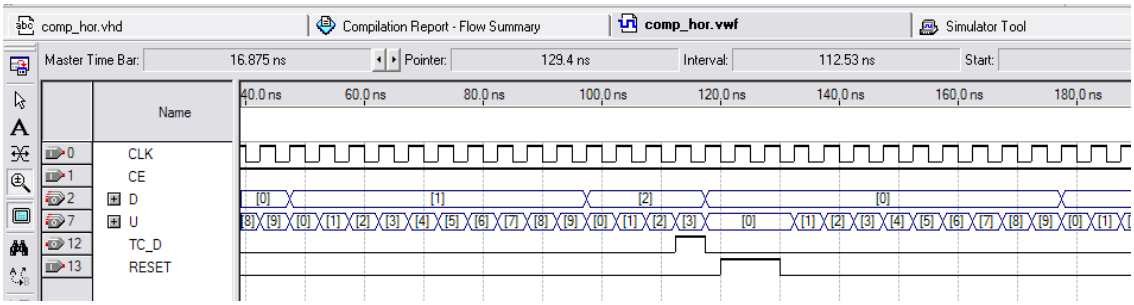


Fig. 2.45. Simulació funcional del comptador d'hores.

2.3.2. Rellotge VHDL. Hores, minuts i segons

En aquest moment tenim totes les eines per construir un rellotge digital. Tenim el comptador de segons que a la vegada ens serveix per crear el de minuts i el comptador d'hores.

Per concretar les senyals que connectaran aquests mòduls veiem el següent diagrama de blocs on veiem les capses, components del rellotge.

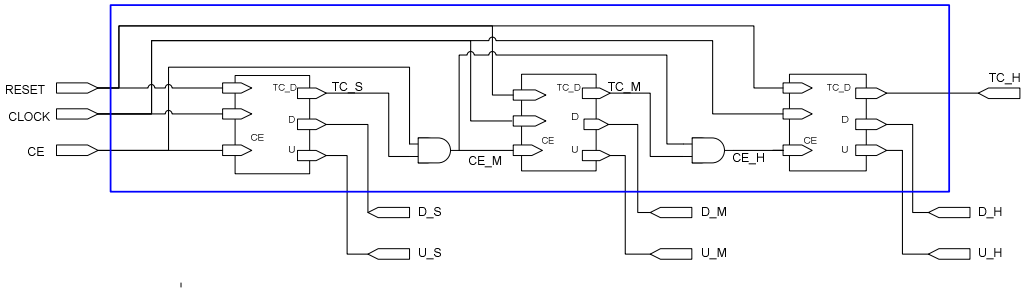


Fig. 2.46. Diagrama de blocs del rellotge VHDL.

Cada una de les capses que apareix a la figura anterior fan referència a cada un dels comptadors que em fet. Fem una petita diferenciació entre entrades i sortides globals del sistema i de les senyals internes. Com a entrades tenim el Reset que l'assignarem a un dels botons de la placa (KEY0), el CE que serà el SW0 i el CLOCK que funcionarà a través de rellotge intern de 27 MHz. Recordem que utilitzarem el divisor de freqüència per aconseguir la freqüència d'1MHz. Com a sortides tenim les unitats i desenes corresponents a segons, minuts i hores. Aquest no els podem dirigir directament a la placa ja que ens és necessari el descodificador de 7 segments que em creat anteriorment. Per una altra banda tenim unes sortides de cada bloc que són les que donaran pas al funcionament del següent. És a dir que quan els segons arribin a '59' el pols que es genera serà l'encarregat de fer sumar la unitat als minuts; de igual manera passarà amb les hores. Tenim que veure que aquesta no és una associació directe ja que si ho fem així quan el minuts augmentin a '59' immediatament les hores també s'incrementarien. Per a que el funcionament sigui el correcte tenim que incorporar unes portes lògiques tal i com veiem a la Fig. 2.46. Aquest últim bloc té el *Terminal count* (TC_H) a massa degut a que no ens cal cap realimentació d'aquesta senyal, tot i que es pot utilitzar per senyalar el final d'un dia si s'associa a un LED de la placa.

Veiem la simulació, com tenen que transcorre 3600 polsos de rellotge per veure el funcionament del comptador d'hores dintre del rellotge fem dues simulacions amb diferents freqüències de rellotge. A la primera veurem la evolució dels segons i minuts i a la segona de tot el conjunt.

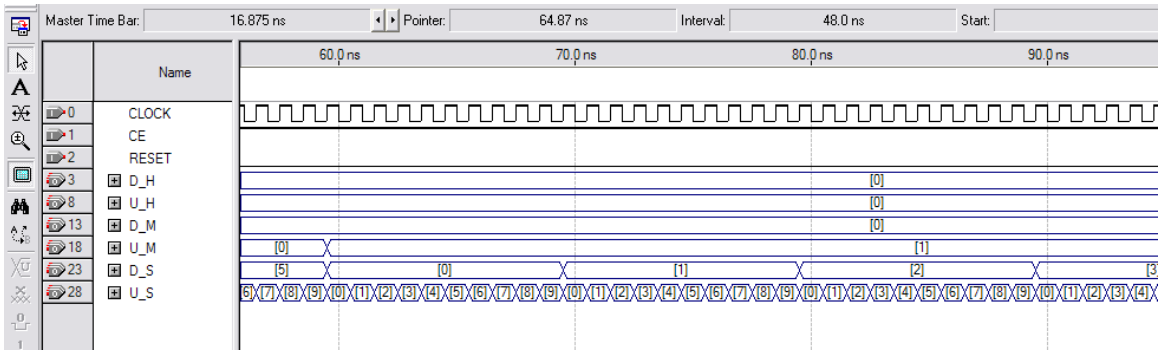


Fig. 2.47. Simulació funcional del rellotge, evolució de segons i minuts.

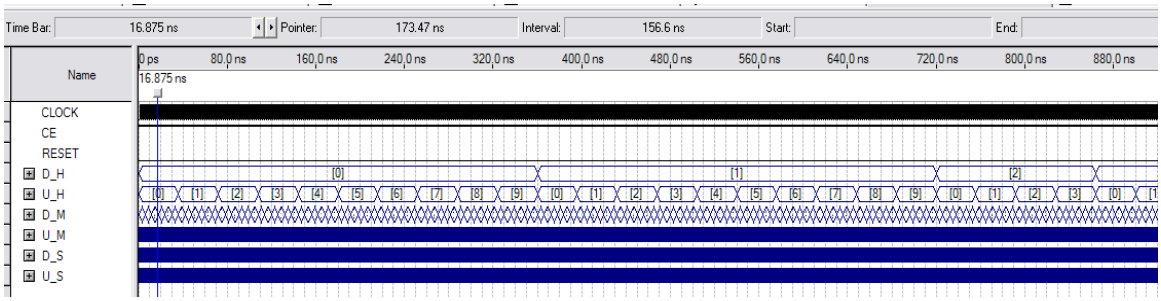


Fig. 2.48. Simulació funcional del rellotge, evolució de les hores.

2.3.3. Ajust de la freqüència i descodificador 7- segments

En aquest moment només ens queda aconseguir que el nostre rellotge sigui autònom, és a dir, ajustar la freqüència d'entrada al *rellotge_digital* a 1 Hz i que podem visualitzar el resultat directament a la placa.

Per això els mòduls adients són el divisor de freqüències (apartat 2.2.1) i el descodificador 7 Segments (apartat 2.2.3).

El que és més important en aquest apartat és fer l'assignació de PINS correctament per a que la placa ens faciliti les eines per iniciar i parar el rellotge quan nosaltres volem. Ja que no hem nombrat les variables tal i com apareixen als manuals d'Altera ho tenim que fer de forma manual.

Tornem a compilar i a carregar l'arxiu *.sof* a la placa. El resultat és el següent:

Fig. 2.49. Resultat final del Rellotge en VHDL

2.3.4. Ajust de l'hora del nostre rellotge

Fins aquest moment posar el nostre rellotge en hora no seria possible si no ho fem a les dotze de la nit prement el *Reset*.

El que ens ocuparà en aquest apartat és fer unes petites modificacions al nostre codi per a que es podem augmentar el valor dels minuts i de les hores quan premem els botons corresponents (KEY1 i KEY2).

El que ens interessa és que s'incrementin hores i minuts però que aquests no s'activin entre ells. Si ens fixem els que ens interessa és modificar el valor de la variable que ens dona pas al funcionament, tant del comptador de minuts com el d'hores. Veiem una possible solució tenint en compte que les dues noves entrades:

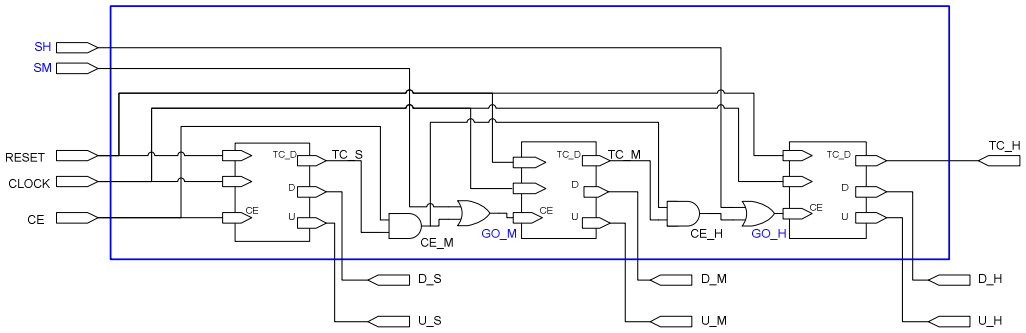


Fig. 2.50. Diagrama de blocs del rellotge amb la funcionalitat de escollir l'hora.

Veiem que hem afegit un parell de entrades noves i dues senyals internes que seran les noves entrades als comptadors d'hores i minuts. Aquests botons (actius a nivell baix) faran que les hores i els minuts augmentin sense necessitat de que el comptador anterior els hi doni pas.

Així només tenim que incorporar al codi del mòdul on enllacem els tres comptadors les modificacions a la *entity* i dues línies de codi on indiquem aquesta nova operació lògica:

```
ENTITY rellotge_digital IS
   PORT (
         RESET, CE, CLOCK ,SM,SH      :IN   std_logic;
         U_S, D_S, U_M, D_M, U_H, D_H :OUT  std_logic_vector(3 downto 0)
      );
END rellotge_digital;

------------------------------------------------------------
--resta de codi

GO_H <='1' WHEN (CE_H='1' OR SH='0') ELSE '0';
GO_M <='1' WHEN (CE_M='1' OR SM='0') ELSE '0';
```

Creem una Màquina d'estats

Amb aquestes modificacions aconseguim posar el nostre rellotge en hora però no hi ha cap regla que el faci para mentre estam manipulant l'hora. Els segons continuen passant mentre posem l'hora correcte. A més, en el cas de que premem tots els botons a l'hora varia tant l'hora com els minuts a una. Ara, el que volem fer és estipular unes normes on s'estableixin prioritats entre les funcionalitats del nostre rellotge.

Veurem com podem construir una Màquina d'estats finits que serà l'encarregada de manipular les entrades per a que la sortida sigui més clara i ordenada. Per veure els estats que tindrà la nostra màquina veiem quines seran les entrades i les sortides que tindrà i com s'enllaçaran entre elles.

Creem la taula de veritat de la FSM (*Finite State Machine*):

CE	SH	SM	GO	GOH	GOM
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	1	0
0	1	1	0	1	0
1	0	0	1	0	0
1	0	1	1	0	0
1	1	0	1	0	0
1	1	1	1	0	0

Fig. 2.51. Taula de veritat de la màquina d'estats.

La variable GO donarà prioritat al funcionament habitual del rellotge, la GOM farà que els minuts es modifiquin i la GOH que ho facin les hores. Hem estipulat que si el rellotge està encès, SW0 activat, no es podrà modificar l'hora. En cas de que aquesta opció estigui apagada i siguin els altres dos botons els que entren en conflicte sempre tindrà prioritat el canvi d'hora sobre el dels minuts.

A partir d'aquesta taula podem veure quantes sortides diferents tindrem, tantes com a estats ens trobarem. Veiem el diagrama d'estats que es dedueix a partir d'aquesta taula:

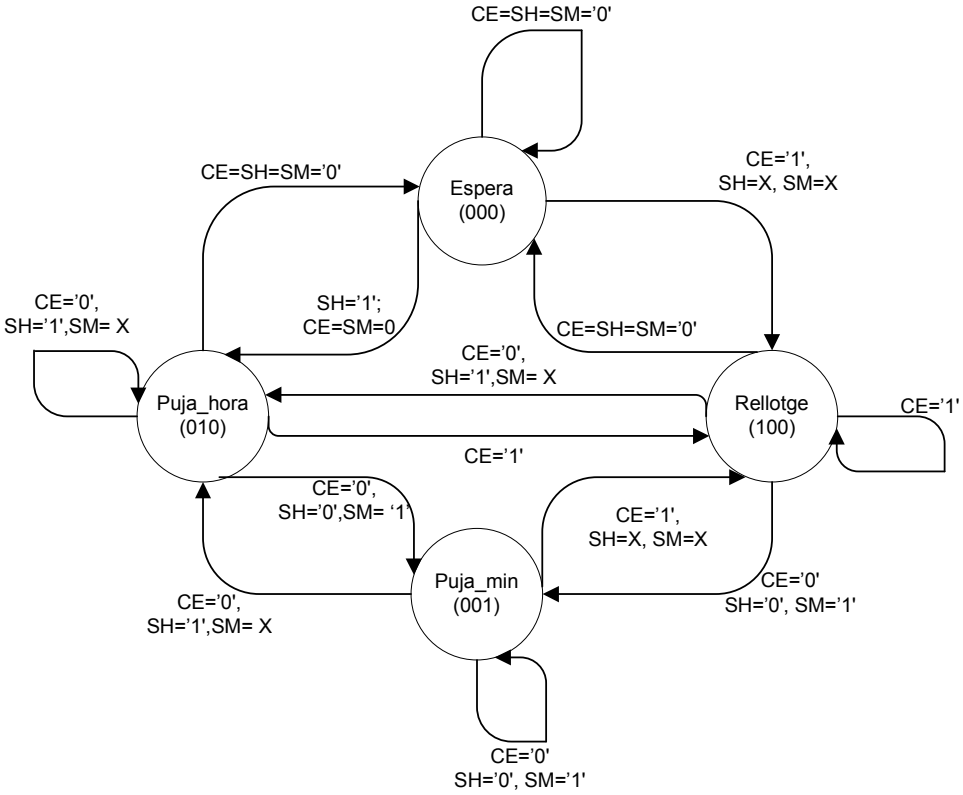


Fig. 2.52. Diagrama d'estats de la FSM.

Al diagrama s'indica els canvis de les entrades que s'han de produir per canviar d'estat. A més dintre de cada estat tenim la sortida corresponent. Així el nostre codi tindrà tres parts diferenciades. La primera en la que es decideix el estat futur depenent de les entrades. La segona on en cada flanc de pujada del clock es fa el canvi d'estat i la tercera on es donarà valor a les sortides depenent del estat on es trobi.

Veiem el codi:

```
ENTITY FSM_RELLOTGE IS
   PORT (
         CLOCK, CE, SM,SH    :IN   STD_LOGIC;
         GO, GOH, GOM  :OUT STD_LOGIC
      );
END FSM_RELLOTGE;
-----------------------------------------------
-----------------------------------------------

ARCHITECTURE FSM_ARCH OF FSM_RELLOTGE IS

TYPE STATE_TYPE IS (ESPERA, RELLOTGE, PUJA_MIN, PUJA_HORA);
SIGNAL PRESENT_STATE, FUTURE_STATE: STATE_TYPE;

BEGIN
   PROCESS(CE, SM,SH, PRESENT_STATE)
   BEGIN
      CASE PRESENT_STATE IS
```

```
   WHEN ESPERA =>
      IF (CE='0' AND SH='1' AND SM='1')THEN
         FUTURE_STATE <= ESPERA;
      ELSIF (CE='1') THEN
         FUTURE_STATE <= RELLOTGE;
         ELSIF (SH='0')THEN
            FUTURE_STATE <= PUJA_HORA;
            ELSIF (SM='0') THEN
               FUTURE_STATE <= PUJA_MIN;
      END IF;

   WHEN RELLOTGE =>
      IF (CE='1') THEN
         FUTURE_STATE <=RELLOTGE;
      ELSIF (SM='0')THEN
         FUTURE_STATE <= PUJA_MIN;
         ELSIF (SH='0') THEN
            FUTURE_STATE <= PUJA_HORA;
         ELSE
            FUTURE_STATE <= ESPERA;
      END IF;

   WHEN PUJA_MIN =>
      IF (CE='1') THEN
         FUTURE_STATE <=RELLOTGE;
      ELSIF (SM='0')THEN
         FUTURE_STATE <= PUJA_MIN;
         ELSIF (SH='0') THEN
            FUTURE_STATE <= PUJA_HORA;
         ELSE
            FUTURE_STATE <= ESPERA;

      END IF;

   WHEN PUJA_HORA =>
         IF (CE='1') THEN
         FUTURE_STATE <=RELLOTGE;
         ELSIF (SH='0')THEN
         FUTURE_STATE <= PUJA_HORA;
         ELSIF (SM='0') THEN
            FUTURE_STATE <= PUJA_MIN;
         ELSE
            FUTURE_STATE <= ESPERA;

      END IF;

   END CASE;
END PROCESS;
-------------------------------------------------
PROCESS (CLOCK)
BEGIN
   IF (CLOCK'EVENT AND CLOCK='1') THEN
      PRESENT_STATE <= FUTURE_STATE;
   END IF;
END PROCESS;
-------------------------------------------------
GO <= '1' WHEN PRESENT_STATE = RELLOTGE ELSE '0';
```

```
    GOH <= '1' WHEN PRESENT_STATE = PUJA_HORA ELSE '0';
    GOM <= '1' WHEN PRESENT_STATE = PUJA_MIN ELSE '0';


END FSM_ARCH;
```

En aquest punt el que ens queda és inserir dintre del projecte final la nostra màquina d'estats. Per fer-ho primer creem el nou diagrama de blocs del sistema final per comprovar quines seran les entrades i sortides de la FSM dintre del nostre projecte.

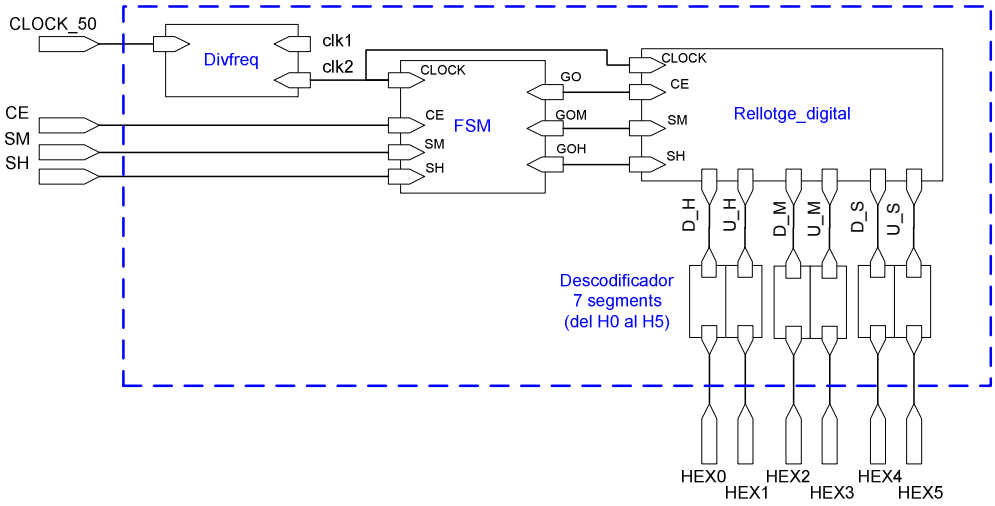


Fig. 2.53. Nou diagrama de blocs del rellotge amb la FSM.

A partir d'aquest diagrama ja veiem que no hi ha noves entrades al sistema però si que tenim que crear noves senyals: GO, GOM i GOH que seran les noves entrades al rellotge digital.
Llavors incorporem un nou COMPONENT al codi VHDL i fem correspondre els ports amb les entrades i senyals corresponents, compilem i carreguem a la placa i veiem que el funcionament és el que esperem.

Fig. 2.54. Rellotge posat en hora.

3. Tutoria sobre NIOS II

En aquest punt el que volem és aprendre a programar el nostre propi microcontrolador. La opció que ens dona el NIOS II és ajustar el nostre propi xip a les necessitats del nostre projecte. Veurem com utilitzar el software necessari per implementar el NIOS II, carregar-lo i programar-lo. Per això Altera ens proporciona una tutoria: *Introduction to the Altera SOPC Builder Using VHDL Design* que la podem trobar a la seva pàgina web [3] o al CD introductori que ens proporcionen amb la placa.

3.1. Construcció d'un projecte amb el nostre propi NIOS II

Comencem per un senzill programa, que com en l'apartat 2.1 ens faci una associació directe entre entrades i sortides del sistema. Encendrem un SW i un LED ens avisarà de que està activat.

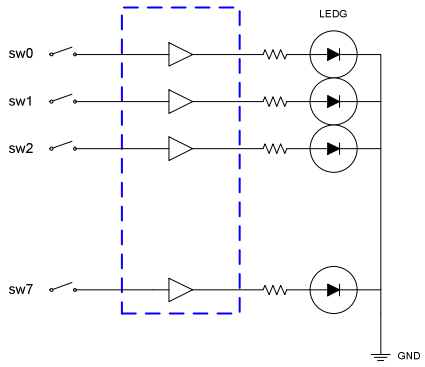


Fig. 3.1 Diagrama de blocs del exercici 1.

3.1.1. Utilització del SOPC Builder: Creació del Hardware

Volem fer una implementació al SOPC Builder per aconseguir tindre un sistema com el que apareix a la Fig. 3.2.

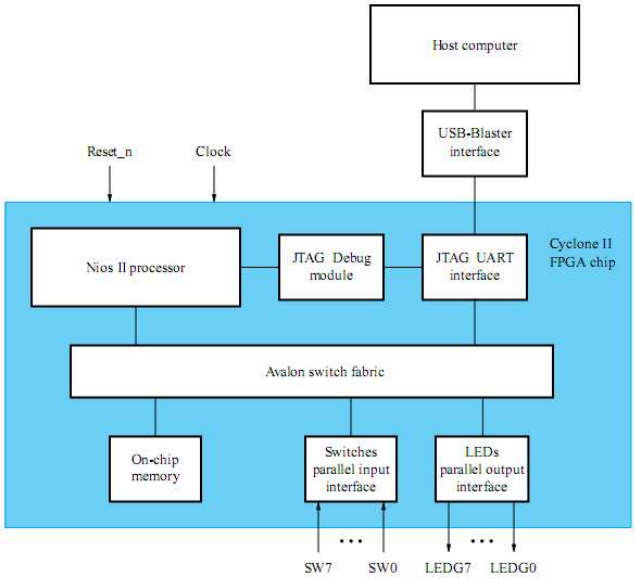


Fig. 3.2. Diagrama de blocs del processador

El que demanarem al programa tindre en el nostre NIOS II serà:

- Un processador NIOS II, equivalent a la CPU.
- Un chip de memòria que escollirem que sigui de 4Kbyte organitzat en paraules de 32 bits.
- Dos interfases I/O paral·leles.
- Una interfase JTAG UART per comunicar-se amb el nostre PC.

Comencem a crear el NIOSII

És necessari crear un nou projecte al *Quartus II* com hem fet fins ara. Li diem lights tant al projecte com a la entitat principal. Escollim el mateix dispositiu que fins ara, EP2C35F672C6.

Per construir el *NIOS* anem a l'eina : *SOPC Builder.*

Tools → SOPC Builder

En aquest moment ens apareixerà la següent pantalla:

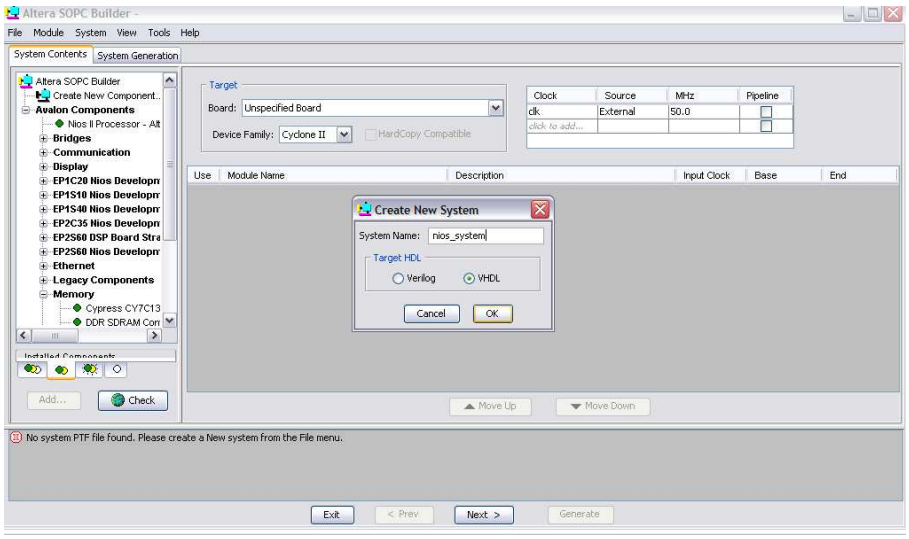


Fig. 3.3 Primera pantalla del SOPC Builder.

El SOPC és l'encarregat de generar un fitxer que faci que *l'Altera Monitor Program* (el veurem en següents apartats) entengui l'arquitectura que nosaltres em creat amb aquest programa. Aquest arxiu és el que ens demana especificar a la finestra que apareix al centre de la pantalla. Li direm '*nios_system'.*

A la pantalla principal del *SOPC* tenim la opció d'escollir la placa i el dispositiu a programar. En el nostre cas tenim una *DE2* que no apareix a la llista, no ens cal escollir cap. En el cas del dispositiu, agafem un *Cyclone II.*

En aquest cas també volem un rellotge a una freqüència de 50 MHz. Aquest ens el proporciona la *DE2*, en aquest cas escollim el nom del rellotge, *clk* i l'especifiquem com a extern. Més tard l'associarem al intern de la placa mitjançant l'associació de PINS.

Per escollir el processador anem a la part esquerra de la pantalla i escollim: *Avalon Components → Nios II processor Altera Corporation*. S'obre una nova pantalla on podem escollir diferents tipus de *NIOS II*. Escollim el més senzill: *NIOS II/e.*

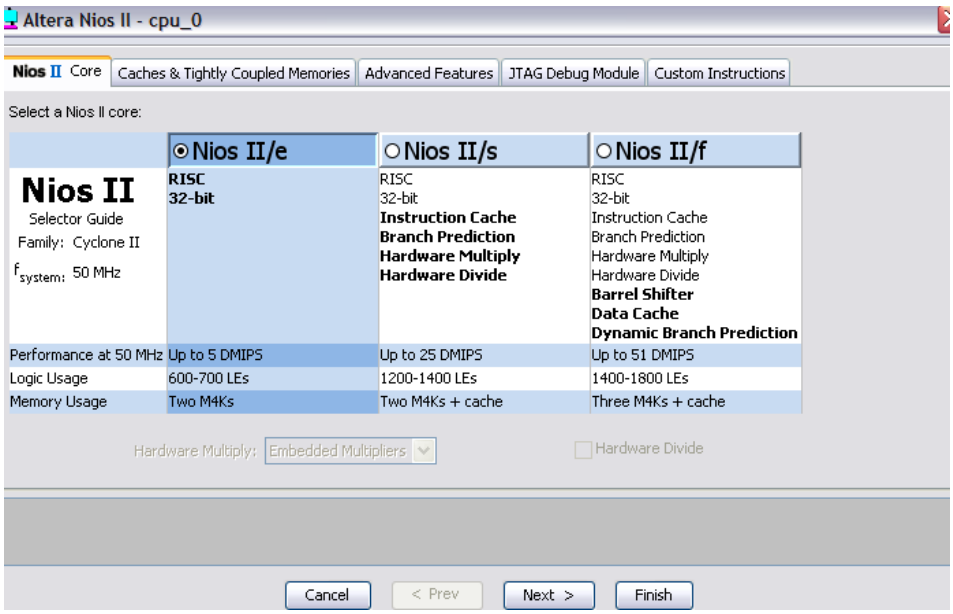


Fig. 3.4 Pantalla de selecció del NIOS II

Els dispositius que anem agregant poc a poc ens van apareixen a la part central de la pantalla. Escollim el xip de memòria: *Avalon Components → Memory → One-Chip Memory (RAM or ROM).* S'obre una nova pantalla on escollim una memòria de 4 Kbytes repartida en paraules de 32 bits.

Especifiquem les entrades i cortides: *Avalon Components → Other → PIO (Parallel I/O).* Cal escollir 8 bits d'entrada i 8 bits (no bidireccionals) de sortida, així primer escollin *Input* i després repetim la operació per les *Outputs*.

Per finalitzar necessitem tindre una interfase per connectar-nos amb el nostre PC, un *JTAG UART*. Així escollim: *Avalon Components → Communication → JTAG UART*. No ens cal modificar cap dels paràmetres que ens venen donats per defecte.

Ja tenim el sistema complet com es pot veure a la Fig. 3.5

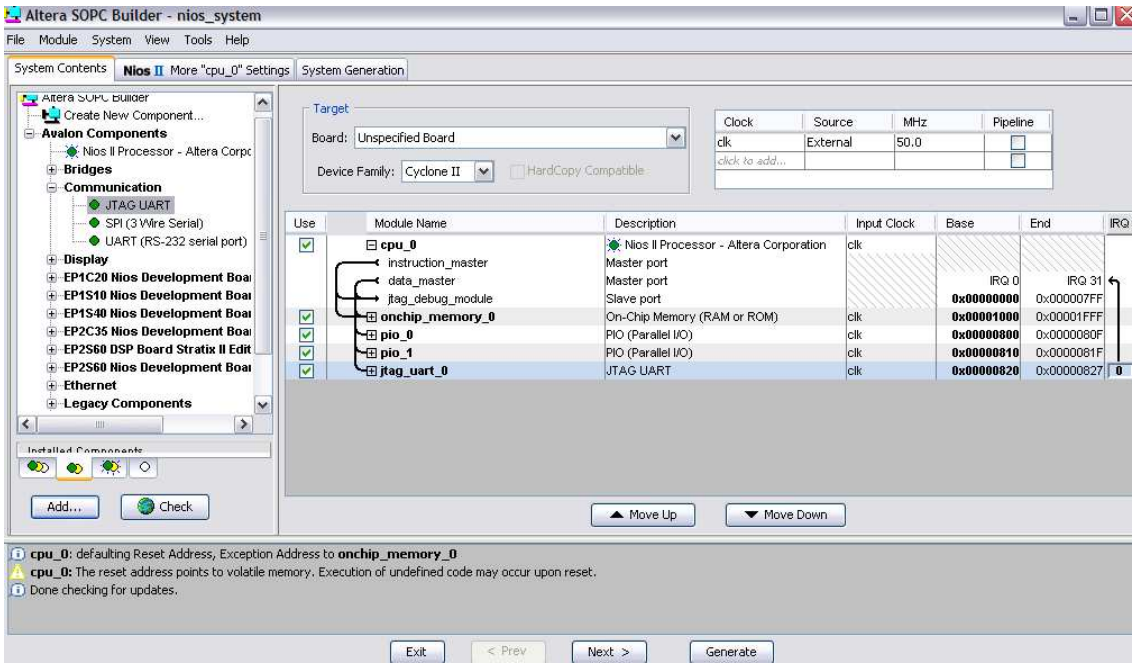


Fig. 3.5. Pantalla final del SOPC Builder

Els noms que SOPC posa als dispositius pot canviar-se per a que els podem relacionar amb l'esquema de la *Fig. 3.2*. Així podem modificar el nom de *pio_0* per *Switches* i el de *pio_1* per *LEDs* prement el botó dret.

SOPC ens dona l'oportunitat de assignar manualment les adreces de memòria o sinó ho pot fer de manera automàtica. Ho farem automàticament: *System → Auto-Assign Base Addresses.* Un cop creades les adreces de memòria tenim que generar el sistema i els arxius pertinents. Per això anem a *System Generator* i desactivem la opció de generar els arxius de simulació ja que no ens calen per aquest projecte. Polsem *Generate*. Si tot és correcte ens apareixerà la pantalla de la Fig. 3.6

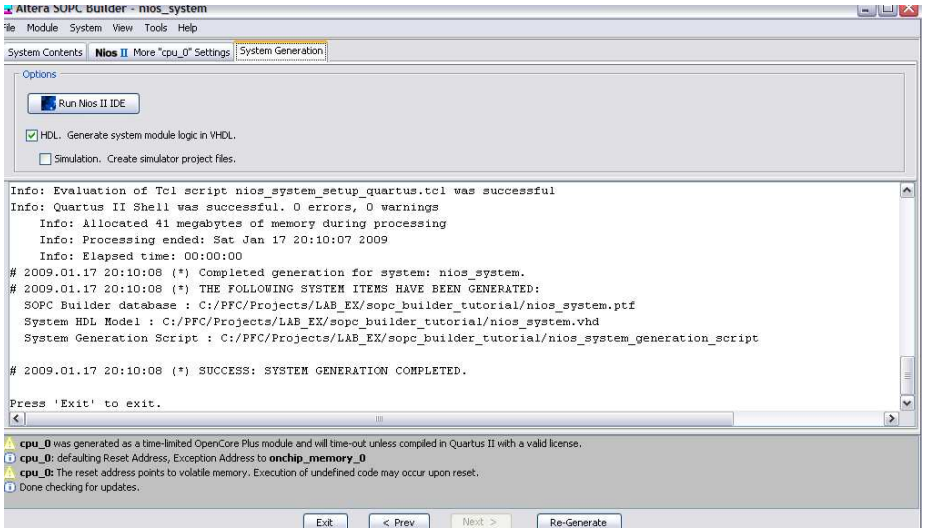


Fig. 3.6. Pantalla final de la generació del nostre processador.

SOPC Builder ens dona l'opció de regenerar el sistema en qualsevol moment, podem obrir el programa, incorporar nous components i tornar a generar.

3.1.2. Integrem el NIOS II dins el projecte de Quartus II

És el moment de incorporar el sistema generat pel *SOPC* al *Quartus II*. Per poder programar el NIOS tenim que assignar els *PINS* de la *FPGA. SOPC* ens ha generat un llistat de fitxers entre els que es troba *el 'nios_system.vhd'.* En aquest fitxer s'especifica la principal entitat del sistema, aquesta és llarga ja que apareixen tots els dispositius que hem incorporat al xip, però Altera ens proporciona un resum:

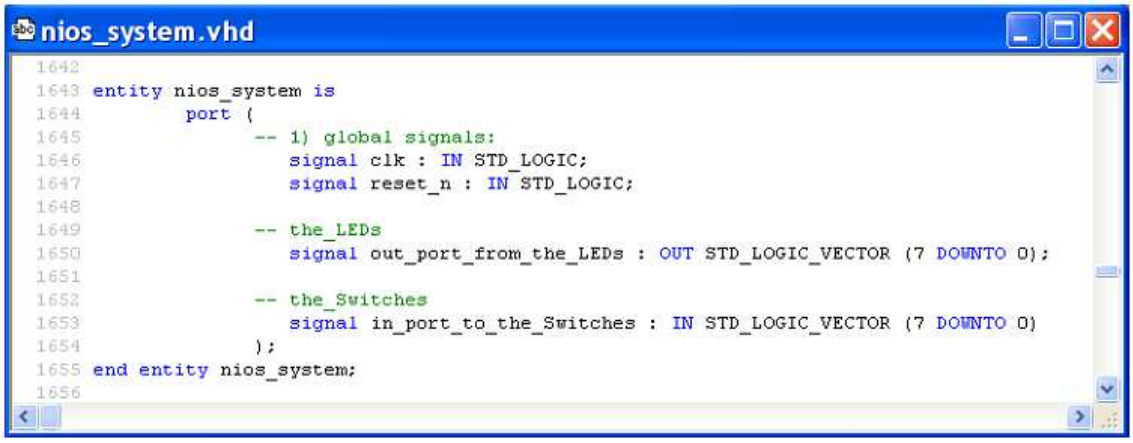


Fig. 3.7. Entitat principal del processador construït.

Veiem que els noms de les variables no coincideixen amb els noms que especifica Altera al fitxer *DE2_pin_assigments.csv* que ens proporciona per fer l'assignació. Farem un nou fitxer *.vhd* per modificar aquests noms i així estalviar-nos fer l'assignació manual. El codi per fer aquesta assignació serà:

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.std_logic_arith.all;
USE ieee.std_logic_unsigned.all;

ENTITY lights IS
PORT (
      SW          : IN STD_LOGIC_VECTOR(7 DOWNTO 0);
      KEY      : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
      CLOCK_50    : IN STD_LOGIC;
      LEDR     : OUT STD_LOGIC_VECTOR(7 DOWNTO 0)
      );
END lights;

ARCHITECTURE Structure OF lights IS
   COMPONENT nios_system
      PORT (
            clk   : IN STD_LOGIC;
```

```
            reset_n : IN STD_LOGIC;
            out_port_from_the_LEDs   :   OUT   STD_LOGIC_VECTOR   (7
DOWNTO 0);
            in_port_to_the_Switches   :   IN   STD_LOGIC_VECTOR   (7
DOWNTO 0)
            );
   END COMPONENT;
BEGIN
   NiosII: nios_system PORT MAP (CLOCK_50, KEY(0), LEDG, SW);
END Structure;
```

Un cop guardat l'arxiu, incorporem al projecte tots els arxius generats per SOPC amb extensió .vhd i l'arxiu *DE2_pin_assigments.csv*

3.1.3. Programa principal: Creació del Software

El nostre processador ha de rebre indicacions de les funcions que té que realitzar.

En aquest cas farem l'aplicació que ja tenim programada en VHDL, una assignació directe entre les entrades dels *Switchs* i les sortides *LEDR*.

En primer lloc farem l'aplicació en Assemblador. Per poder entendre el codi, Altera ens proporciona un manual sobre el NIOS II i com programar-lo en aquest llenguatge [5]

Veiem el codi per aquesta aplicació:

```
.include   "nios_macros.s"
.equ    Switches, 0x00001800
.equ    LEDs, 0x00001810
.global _start

_start:
   movia r2,   Switches
   movia r3,   LEDs

loop:
      ldbio r4,   0(r2)
   stbio r4,   0(r3)
   br loop
```

Al inici del programa indiquem la llibreria on es troben les especificacions de les macros que utilitzem a l'aplicació com per exemple la comanda movia que ens ve especificada dins d'aquesta. Les variables *(Switches, LEDs)* també les creem abans de donar inici al programa principal; per indicar que comença utilitzem la sentencia ***_start.***

Per carregar el programa a la placa creem un fitxer amb nom *lights.s* el qual serà el que indiquem al programa que carregui a la *DE2*. Per crear-ho utilitzem l'aplicació de '*Bloc de Notas'* de *Windows* i guardem el fitxer amb l'extensió indicada.

Una altra forma de programar el nostre processador és en llenguatge C, el qual és més instintiu i més fàcil d'aprendre. L'estructura fonamental del programa en C és semblant a la d'assemblador, consta de la declaració de llibreries en primer lloc i de la declaració de variables per continuar pel cos del programa introduït per la sentència *void main()*. En aquest cas no tenim cap manual que ens especifiqui les sentències que podem utilitzar ni l'estructura que es té que fer servir. Per això es poden trobar molts llibres sobre programació en C.

Per poder carregar a la placa el nostre programa tenim que crear un fitxer amb el nom de *lights.c* que serà el que indiquem al *Altera Monitor Program* que té que compilar i carregar a la *DE2.*

3.1.4. Càrrega del programa principal: Altera Monitor Program

Ara ja tenim el processador que hem fet carregat a la placa. Tenim que fer el programa amb l'aplicació final i carregar-ho a la DE2. Per això Altera ens dona diferents opcions, com hem vist anteriorment farem servir el *Altera Monitor Program*, el qual compila i carrega el programa a la placa sense que nosaltres tinguem que fer res més que prémer el botó corresponent.

En primer lloc tenim que dir al programa quin és el nostre processador. Per això el *Quartus* ens ha creat un fitxer *.ptf*. Aquest porta la informació dels dispositius que hem escollit i de la memòria que necessitarem. Per carregar-ho anem al menú: *Configuration → Configure System* o al primer botó de la segona barra de la part superior de la pantalla.

A la finestra que apareix tenim que indicar l'arxiu .ptf del nostre projecte a més del cable que utilitzem per comunicar-nos amb la placa. La informació es carrega de manera automàtica quan indiquem el nom del fitxer. Al final ens té que quedar la pantalla tal i com s'indica a la Fig. 3.8. Confirmem la informació prenent *ok.*

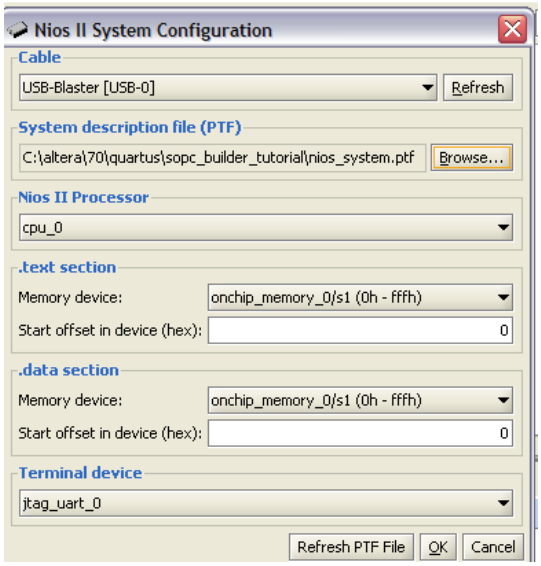


Fig. 3.8. Pantalla de configuració del sistema.

En segon lloc, observem que un altre icona s'ha il·luminat a la barra superior de la finestra principal. És la corresponent al menú: *Configuration → Configure program.* Aquí indicarem el fitxer del programa principal. Com ja hem vist, tenim l'opció de fer-ho en diferents llenguatges de programació.

Si anem al menú per carregar el programa ens apareix la pantalla de la Fig. 3.9. Aquí indiquem el llenguatge utilitzat, assemblador, i el fitxer *.s* que prèviament hem creat. Recordem que per crear el '*lights.s'* hem obert el '*Bloc de Notas'* i hem guardat l'arxiu amb l'extensió indicada. El *Start Symbol* és l'indicatiu que el programa assimilarà com a inici del nostre programa.

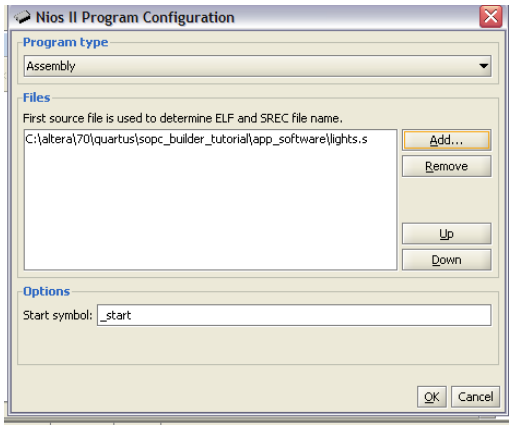


Fig. 3.9. Pantalla de configuració del programa.

Per compilar i carregar el programa principal tenim diferents opcions, podem demanar al programa que primer compili i després li demanem que carregui o directament que ho faci tot seguit sense esperar les nostres indicacions. L'Altera Monitor en tot moment ens dona informació de l'estat en que es troba.

La càrrega es pararà on hem indicat el inici del programa (*_start*), tal i com es veu a la Fig. 3.10

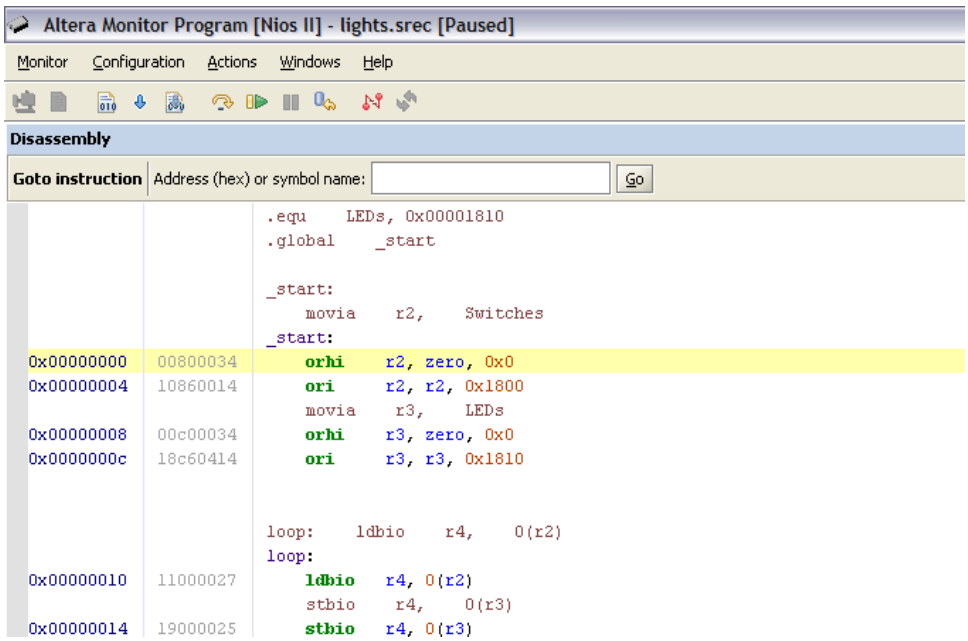


Fig. 3.10. Pantalla principal del Monitor Altera en procés de càrrega del programa.

Per a que continuem amb el programa podem anar al menú: *Actions* → *Continue*. A partir d'ara veurem com la placa ja té el nostre programa guardat. En qualsevol moment tenim la possibilitat de parar l'execució del programa anat a *Actions* → *Stop*. A més podem incorporar *Break Points* dintre del programa i aquest s'encarregarà d'aturar l'execució automàticament.

Per més detalls es pot consultar la tutoria que ens ofereix Altera a la seva web [1].

Podem carregar el fitxer .c enlloc del .s si indiquem que el codi utilitzat és C i no assemblador. El resultat final en tots dos casos és el que podem veure a la Fig. 3.11.

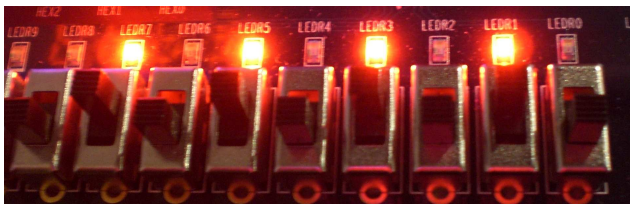


Fig. 3.11. Vista final del projecte.

3.2. Interrupcions

En aquest apartat veurem un petit exemple amb interrupcions. A partir d'aquí aprendrem a manipular-les: habilitar-les,inhabilitar-les i complementar el codi que executaran quan el programa salti al estat interrupció.

3.2.1. Creació del NIOS tenint en compte les interrupcions

En aquest apartat utilitzarem una estructura bàsica de NIOS que Altera ens proporciona amb el programa: Altera Monitor Program. Veurem com funcionen les interrupcions i les prioritats que apareixen entre elles amb un petit programa en C.

El programa principal esperarà a que la KEY3 es premi per a canviar l'estat dels leds que continuaran lligats a la posició dels interruptors. És a dir, si tenim activat un interruptor, el led corresponent no s'encendrà fins que no s'accioni la KEY3. Igualment passarà si volem que s'apagui, fins que el botó no s'accioni per molt que s'hagi desactivat l'interruptor no s'apagarà.

En primer lloc tenim que localitzar els fitxers que Altera ens dóna, els tenim situats a: *C:\altera\nios2eds\bin\monitor\samples.* Aquest adreça pot canviar segons on haguem instal·lat els nostres arxius.

Basic Computer

Ens basarem en la màquina bàsica que Altera ens dona creada. Per carregarla només tenim que obrir el projecte de Quartus següent: *C:\altera\nios2eds\bin\monitor\samples\Systems\DE2_Basic_Computer\DE2_Basic_Computer.qpf*. Aquest programa ja el tenim compilat i llest per carregar a la placa. Com és habitual, el fabricant ens proporciona un manual [7] per a que fem servir aquesta màquina correctament on podem trobar el següent diagrama de blocs de la màquina bàsica:

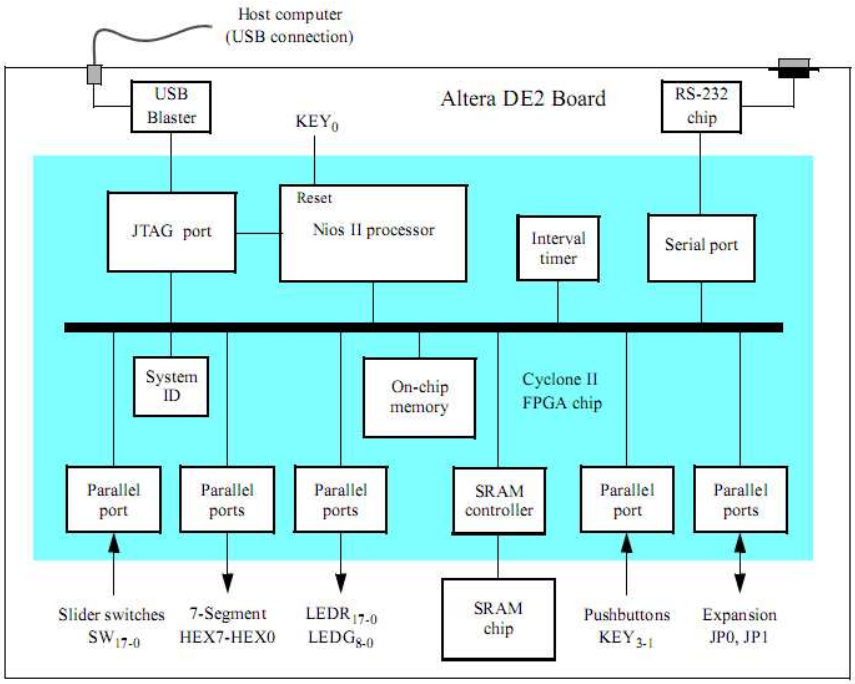


Fig. 3.12. Diagrama de blocs de la 'Basic Computer' d'Altera.

Una altra manera de saber quins són els dispositius que tenim a la nostra disposició si fem servir aquest sistema és obrir el projecte en Quartus i consultar el SOPC Builder. Aquí veiem més informació que ens serà útil per després treballar, com ara: rangs de memòria, interrupcions, etc.

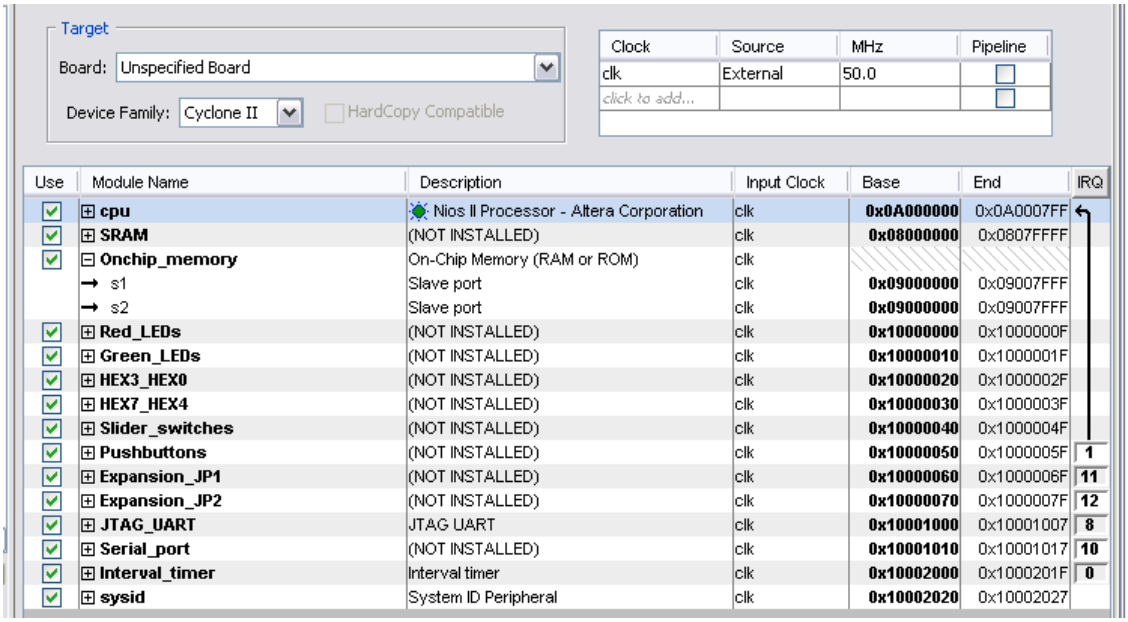


Fig. 3.13. Dispositius inclosos dintre de la Màquina bàsica d'Altera.

Veiem que hi tenim dispositius que no utilitzarem ja que el nostre programa només tindrà les entrades dels interruptors (*Slider_switches*) i la dels botons (*Pushbuttons*). Les sortides seran els leds vermells (*RED_LEDs*).

És important que les interrupcions estiguin bé definides com és aquest cas. Veiem a l'última columna com les interrupcions estan numerades, aquesta numeració ens indica l'ordre de prioritat (la més baixa té prioritat més alta). En el nostre cas volem que les interrupcions ens les provoquin els botons, per això quan tinguem una interrupció tindrem que verificar que aquesta és la identificada amb el número 1 (IRQ1).

Com hem anat fent fins ara tenim que carregar l'arxiu .sof, que ja tenim creat, a la placa per a que ens pugui reconèixer el nostre microcontrolador.

3.2.2. Creació del programa principal. Interrupcions dels botons

En la carpeta: *C:\altera\nios2eds\bin\monitor\samples\Programs\pushbutton_interrupts* trobarem els arxius en C que utilitzarem per aquest petit projecte. Tenim dos fitxers:

- *isr_linkage.*c : Aquest fitxer ens proporciona les eines per poder utilitzar les interrupcions. Quan volem fer qualsevol altre projecte en el que necessitem crear interrupcions sempre el podrem incloure.

- *Pushbuttons_interrupt.c*: Aquest fitxer conté el programa principal i les funcions que seran executades quan tinguem una interrupció. Veiem i expliquem el seu contingut:

```
#define SWITCHES_BASE_ADDRESS  0x10000040
#define LEDR_BASE_ADDRESS    0x10000000
#define PUSHBUTTONS_BASE_ADDRESS 0x10000050

void interrupt_handler(void)

/*****************************************************************/
/* Interrupt Service Routine                                     */
/*   Determines what caused the interrupt and calls the appropriate */
/*  subroutine.                                                  */
/*                                                               */
/* ipending - Control register 4 which has the pending external     */
/* interrupts                                                    */
/*****************************************************************/

{
   int ipending;
   ipending=__builtin_rdctl(4);//Read the ipending register
   if ((ipending & 0x2) == 2) //If irq1 is high, run pushbutton_isr,
otherwise return
   {
      pushbutton_isr();
   }
```

```
   return;
}

void pushbutton_isr(void)
{
   int * red_leds = (int *) LEDR_BASE_ADDRESS;
   int* pushbuttons = (int *) PUSHBUTTONS_BASE_ADDRESS;
   volatile int * switches = (int *) SWITCHES_BASE_ADDRESS;

   *(red_leds)=*(switches);//Make LEDs light up to match switches

   *(pushbuttons+3)=0; //Disable the interrupt by writing to
edgecatpure registers of the pushbuttons

   return;
}

int main(void)
/*****************************************************************/
/* Main Program                                                  */
/*   Enables interrupts then loops infinitely                    */
/*****************************************************************/
{
   volatile int * switches = (int *) SWITCHES_BASE_ADDRESS;
   volatile int * pushbuttons= (int *) PUSHBUTTONS_BASE_ADDRESS;
   int * red_leds = (int *) LEDR_BASE_ADDRESS;

   *(pushbuttons+2)=0x8; //Enable KEY3 to enable interrupts

   __builtin_wrctl(3, 2);//Write 2 into ienable register
   __builtin_wrctl(0, 1);//Write 1 into status register
   while(1);
   return 0;
```

Al inici del programa tenim la declaració de diferents constants que fan referència a les adreces de memòria de les entrades i de les sortides. A partir d'aquestes adreces, afegint-li els offsets pertinents veurem com activar i desactivar les interrupcions.

És en el programa principal on habilitem les interrupcions corresponents als botons (IRQ0). Per fer-ho posem a '1' el bit corresponent a aquesta IRQ del registre *ienable* (Fig. 3.14). A més si posem el bit '0' del registre *status* (PIE) habilitarem totes les interrupcions.

La funció: *void interrupt_handler(void)* és l'encarregada de testejar per qui ve provocada la interrupció. Automàticament, quan hi ha una interrupció en el programa, sense necessitat de cridar a aquesta funció al programa principal, s'executarà i ens donarà la sortida que haguem demanat. En aquest cas verifica que la causa de la interrupció sigui el botó 3: *(ipending & 0x2) == 2*. Si mirem els manuals (Fig. 3.14,[7]) trobem que el registre *ipending* posa els seus bits actius si una interrupció està activa. A la taula de SOPC Builder veiem que els botons activen la IRQ1, per això s'aplica una màscara a 0x2 i es verifica que en segon bit sigui un '1' (2 en decimal equival a 10 en binari).

Veiem la taula dels registres de les interrupcions:

Register	Name	31...1	0
ctl0	status	*Reserved*	PIE
ctl1	estatus	*Reserved*	EPIE
ctl2	bstatus	*Reserved*	BPIE
ctl3	ienable	Interrupt-enable bits	
ctl4	ipending	Pending-interrupt bits	
ctl5	cpuid	Unique processor identifier	

Fig. 3.14. Registres afectats per les interrupcions.

Si aquesta condició es compleix, el programa ens envia cap a la següent funció: *void pushbutton_isr(void)*. Aquesta s'encarrega de fer l'assignació entre els interruptors i els leds. Veiem que després posa a '0' el valor del bit *edgecatpure*. Per entendre aquesta assignació veiem el quadre que trobem al *datasheet* de les PIO [9] que ens proporciona el fabricant:

A1..A0	Register Name		R/W	Variable Size—1 to 32 bits
0	data	read	RO	Data value currently on PIO inputs
		write	WO	New value to drive on PIO outputs
1	direction		RW	Data direction (optional): Individual control for each PIO bit
2	interruptmask		RW	Interrupt mask (optional): Per-bit IRQ enable/disable
3	edgecapture[1]		RW	Edge capture (optional): Per-bit synchronous edge detect and hold

Fig. 3.15. Mapa de bits de les entrades i sortides.

En aquest quan tinguem una interrupció el bit *edgecapture* el tindrem a '1', per aquesta raó quan acabem de fer les feines que aquesta interrupció implica tornem a posar aquest bit a '0': **(pushbuttons+3)=0;*.

Un cop el programa deixa d'executar la interrupció torna al programa principal. En aquest cas veiem que en aquesta part de programa l'únic que es fa és activar les interrupcions provocades pel botó *(*(pushbuttons+2)=0x8, interruptmask* a '1') i un *while* sense fi.

Un cop ja sabem el que el codi genera, carreguem l' Altera Monitor Program. Com a sistema obrim l'arxiu que trobem dintre de la mateixa carpeta que el projecte de Quartus amb el nom: *nios_system.ptf* i incorporem un offset tant per la memòria de programa com a la de dades per a que les interrupcions es puguin executar, tal i com apareix a la Fig. 3.16:

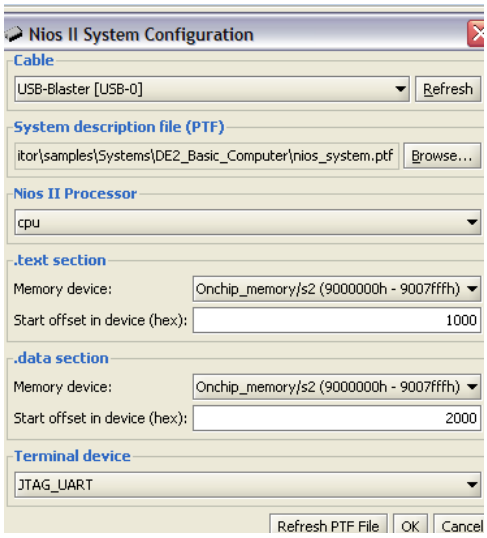


Fig. 3.16. Càrrega del sistema creat al Altera Monitor program.

A l'hora de carregar els programes en C ho fem de manera que l'arxiu principal: *pushbutton_interrupts.c* quedi a dalt de tot com veiem a la figura:

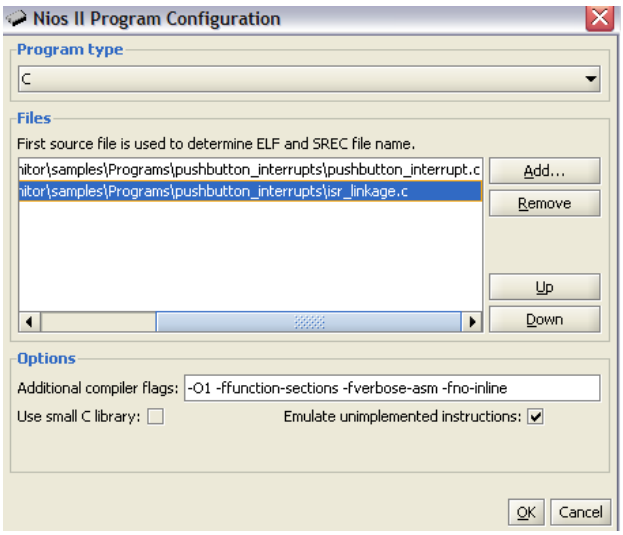


Fig. 3.17. Càrrega dels programes en C al Altera Monitor Program.

En aquest moment només queda compilar i carregar el programa com hem vist en apartats anteriors que hem fet amb el *Monitor Program*. El resultat final és el següent:

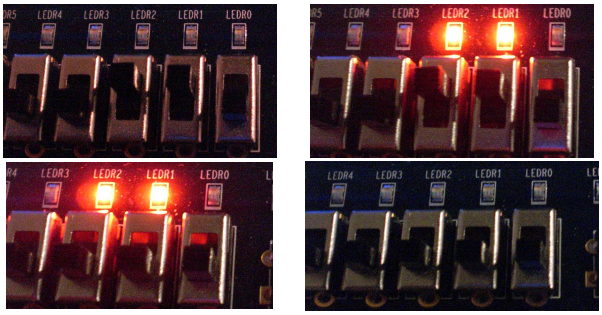


Fig. 3.18. Diferents fases del projecte final.

3.2.3. Manipulació del programa principal. Interrupcions del TIMER

Com hem vist a la Fig. 3.12, la màquina bàsica ens proporciona un TIMER intern que podem utilitzar per a que ens causi interrupcions de manera periòdica. El SOPC Builder ens deixa crear aquest TIMER com a nosaltres ens vagi millor. Per saber una mica més sobre aquest podem consultar el Datasheet d'aquest dispositiu [10][11].

TIMER

En la màquina bàsica proporcionada per Altera tenim un TIMER intern que ens proporciona una interrupció cada 125 ms. Aquesta informació la trobem a la tutoria d'Altera al respecte d'aquesta màquina [7].

El que ara volem és manipular aquest Timer per a que s'encarregui de comptar els temps real i que el programa principal executi unes funcions determinades en aquest temps. Per això necessitem saber activar i desactivar les interrupcions. Per aquest motiu necessitem entendre l'arquitectura interna d'aquest dispositiu. Veiem el diagrama de blocs:

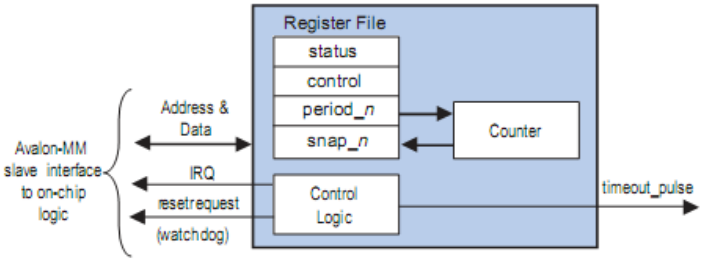


Fig. 3.19. Diagrama de blocs del timer intern del NIOS II.

Veiem que tenim diferents registres i cada un d'ells conté una informació determinada que farà que el Timer funcioni d'una determinada manera. Veiem el mapa dels bits, ens fixem en els dos que tindrem que manipular per adaptar-ho a les nostres necessitats: *status* i *control*.

Offset	Name	R/W	Description of Bits						
			15	...	4	3	2	1	0
0	status	RW	(1)					RUN	TO
1	control	RW	(1)			STOP	START	CONT	ITO
2	periodl	RW	Timeout Period – 1 (bits [15:0])						
3	periodh	RW	Timeout Period – 1 (bits [31:16])						
4	snapl	RW	Counter Snapshot (bits [15:0])						
5	snaph	RW	Counter Snapshot (bits [31:16])						

Fig. 3.20. Mapa de registres del TIMER.

Fem una explicació breu de cada un dels bits més significatius:

- ITO: bit 0 del registre *control*. Aquest bit ens activem les interrupcions del Timer, si aquest no està actiu (‘1’) el Timer no cousarà cap interrupció.

- START: bit 2 del registre *control*. Aquest bit farà que el Timer avanci. Si aquest bit està a ‘0’ el Timer no contarà i per lo tant serà com si aquest no estigués al sistema. El podem utilitzar per encendre el Timer en un moment determinat del programa.

- CONT, STOP: bit 1 i 3 respectivament del registre *control*. Si CONT està actiu, el Timer compta contínuament fins que el programa el para posant a ‘1’ el bit STOP. En cas de que CONT sigui ‘0’ aquest comptarà contínuament fins arribar a zero, després es carregarà automàticament amb el valor inicial de Timer que haguem indicat.

- TO: bit 0 del registre *status*. Aquest és posarà a ‘1’ quan el Timer hagi causat una interrupció. Quan sortim d’aquesta interrupció necessitem posar aquest bit a ‘0’ de nou.

- RUN: bit 1 del registre *status*. Aquest registre es posa a ‘1’ quan el Timer està en marxa. Només és de lectura, no el podem activar i desactivar manualment.

Nova interrupció al programa principal

En l‘apartat anterior (3.2.2) hem vist com activar les interrupcions que venien donades per el botó. En aquest cas tenim que modificar el codi per a que també ens deixi fer interrupcions provinents del Timer. Per això canviarem els següents registres:

- Registre *ienable* (32 bits)*:* Cada un dels seus bits activa una interrupció. El Timer té associada la interrupció IRQ0. Per aquest motiu li donarem un valor de ‘3’ (Deixem actives les interrupcions dels botons, IRQ1).

```
__builtin_wrctl(3, 0x3);
```

- Registre control (16 bits)del Timer: Activem les interrupcions del Timer i fem córrer el comptador. Bit ITO, START i CONT a ‘1’.

```
int *timer =(int *) TIMER_BASE_ADDRESS;
            *(timer+1)=0x7;
```

Si declarem una nova constant amb l’adreça de memòria del Timer podem utilitzar aquesta com a base per a manipular els registres interns com hem fet amb la resta de dispositius a l’apartat anterior.

Amb aquests canvis pel programa ens causarà una interrupció cada 125 ms. Aquesta interrupció cridarà directament la funció: *void interrupt_handler(void)* que s'encarregarà de determinar de quin dels dos dispositius que tenen les interrupcions actives ha provocat aquesta crida. Llegeix el valor del registre *ipending*. Depenent del resultat saltem a la funció corresponent. En cas de que l'hagi causat el Timer hem creat la següent funció:

```
void timer_isr(void)
{
   int * red_leds = (int *) LEDR_BASE_ADDRESS;
   int *timer = (int *)TIMER_BASE_ADDRESS;

   if (interrupcio>4)
      *(red_leds)= 0x3ffff;
   else
      *(red_leds)= 0x0;


   if(interrupcio==8)
         interrupcio=0;
   else
      interrupcio++;


   *(timer)=0x0;
}
```

Veiem que hem creat una variable que fa de comptador d'interrupcions, aquesta la utilitzem per a que els leds s'encenguin un cop per segon. Com tenim una interrupció cada 125 ms al segon en tenim vuit. Així tenim durant 4 interrupcions els leds encesos i durant 4 més apagats. Sempre que sortim de la interrupció tornem el bit TO a '0'.

Si carreguem la *Basic Computer* a la DE2 i després amb l'ajuda del Altera Monitor program compilem i carreguem el nou programa comprovem que els leds s'encenen i apaguen un cop per segon.

3.3. Displays 7-Segments

En aquest apartat volem començar a treballar amb els Displays 7 segments. Modificarem el codi anterior per veure com podem fer un comptador del '0' al '9' que s'incrementi constantment cada segon. Per això continuarem utilitzant la interrupció del Timer.

3.3.1. Descodificador 7-Segments

Per tal de que el resultat que veiem a la placa sigui el correcte necessitem fer un descodificador de decimal a 7-segments tal i com vam fer amb VHDL (Apartat 2.2.3). En aquest cas veurem que serà molt més fàcil.

Tindrem un comptador que cada cop que Timer causi vuit interrupcions (recordem que tenim un període de 125ms que seran vuit interrupcions per segon) s'incrementarà i ens donarà el valor en decimal. Si ho fem així, podem crear un vector amb cada una de les traduccions a 7-segments i que aquest comptador sigui l'índex d'aquest. Veiem la taula de traduccions i el codi de la funció en qüestió:

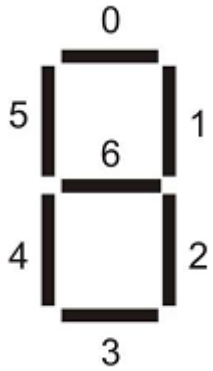


Fig. 3.21. Distribució dels bits del 7-segments

Dígit	HEX0	7-Seg
0	0111111	0x3f
1	0000110	0x06
2	1011011	0x5b
3	1001111	0x4f
4	1100110	0x66
5	1101101	0x6d
6	1111101	0x7d
7	0000111	0x07
8	1111111	0x7f
9	1100111	0x67

Fig. 3.22. Taula de veritat del descodificador.

```
char descodifica_bcd_7segments (int index)
{
   char hex[11]={0x3f, 0x06, 0x5b, 0x4f, 0x66, 0x6d, 0x7d, 0x07, 0x7f,
0x67};

   return hex[index];
}
```

Amb aquesta funció el que farem serà anar traduint el valor del comptador mentre el programa principal espera que alguna de les interrupcions pari la seva execució. Veiem el nou codi de la interrupció del Timer:

```
void timer_isr(void)
{
   int * red_leds = (int *) LEDR_BASE_ADDRESS;
   int *timer = (int *)TIMER_BASE_ADDRESS;
   if (interrupcio==8)
   {
      interrupcio=0;
      if(comptador<9)
         comptador++;
```

```
      else
         comptador=0;
   }
   else
      interrupcio++;

   *(timer)=0x0;
}
```

i el nou *while* del programa principal:

```
while(1){
   *(hex)= *(int *)descodifica_bcd_7segments(comptador);
   };
```

Sent *hex*, com en els casos anteriors un punter a la adreça de memòria base que hem declarat al principi del programa com una contant:

```
#define  HEX_BASE_ADDRESS 0x10000020
volatile int *hex =(int *) HEX_BASE_ADDRESS;
```

Si carreguem aquest codi a la placa amb la Basic Computer com a Hardware veurem com al HEX0 van apareixen els dígits consecutius cada segon.

Fig. 3.23. Visualització final del projecte.

Utilització de més d'un 7-Segments

Si consultem el manual de la Màquina Bàsica d'Altera podem veure com tenim tots els Displays distribuïts en memòria. Veiem el quadre explicatiu que ens proporcionen:

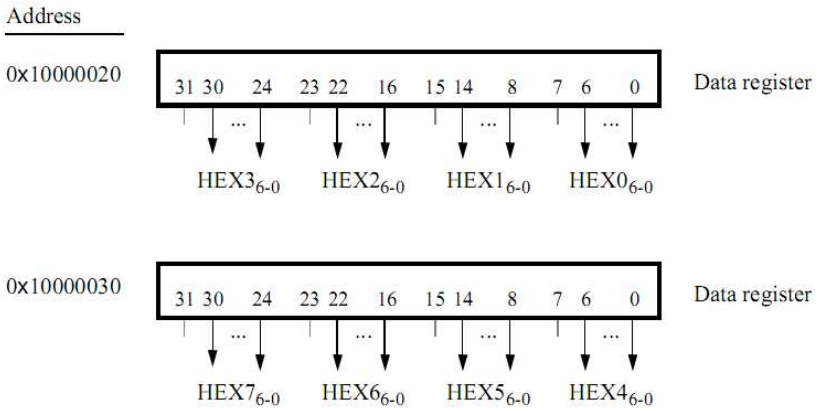


Fig. 3.24. Distribució de la memòria per els Displays 7-Segments.

Tenim dos adreces base de les quals pengen 4 Displays. Cada 8 bits tenim els registres d'un 7-Segments.

Per veure com podem posar els resultats en aquests registres farem un exemple senzill que consti en posar el mateix resultat del comptador en els 4 primers displays.

Si profunditzem en el llenguatge C sabrem que cada tipus de variable que podem declarar té un determinat número de bits. En aquests exemples veiem que majoritàriament estem treballant amb enters i caràcters. Cada un d'ells té una capacitat de 32 i 8 bits respectivament. Per això per omplir els registres dels diferents Displays utilitzarem combinacions d'aquest dos tipus de variables. Veiem el codi i la seva explicació:

```
int void main()
{
/*****************************************************************
...declaració de més variables
*****************************************************************/
   volatile int *hex =(int *) HEX_BASE_ADDRESS;
   char hex_segments[]={0,0,0,0,0,0,0,0};
   int i;
   interrupcio=0;
   comptador=0;

/*****************************************************************
Activació de les interrupcions
*****************************************************************/

while(1)
   {
      hex_segments[0]= descodifica_bcd_7segments (comptador);
      for(i=0; i<8;i++)
      {
         hex_segments[i]=hex_segments[0];
      }
      *(hex)=*(int *)hex_segments;
   };
   return 0;
}
```

Declarem un vector de vuit de *char*. Com hem dit abans, un *char* equival a un byte, si calculem, tenim 8x8 = 64 bits, que seran els 64 bits que ens ocupen els 8 displays. En aquest exemple només utilitzem els 4 primers (32 bits).

Quan li donem el valor d'aquest vector, convertit a enter, al registre que tenim apuntant a la memòria dels 7 segments (*hex*), no fem res més que comunicar-li que el valor dels 32 bits estan situats als 32 bits a partir de la memòria base que li hagi assignat el programa al vector on guardem la informació de la descodificació (*hex_segments*). A partir d'aquí comprovem el resultat a la placa:

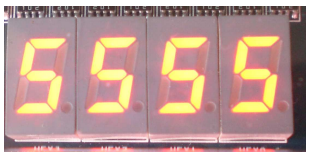


Fig. 3.25. Visualització final del projecte.

Si volem incorporar la resta de 7 segments al nostre projecte només tenim que declarar la adreça corresponent com hem fet amb tots els dispositius i donar-li el valor dintre del while com hem fet amb els quatre primers. Veiem la solució:

```
while(1){
   hex_segments[0]= descodifica_bcd_7segments (comptador);
   for(i=0; i<8;i++){
      hex_segments[i]=hex_segments[0];
      }
   *(hex)=*(int *)hex_segments;
   *(hex_2)=*(int *)(hex_segments+4);
   };
```

On hex_2 és un enter que apunta a la posició base de memòria dels quatre 7-segments que ens quedaven per testejar.

Fig. 3.26. Visualització final del projecte.

3.4. Rellotge digital

En aquest apartat anirem construint pas a pas un rellotge digital com ja hem fet anteriorment amb VHDL. Veurem la simplicitat que ens proporciona tenir el nostre propi NIOS i poder-ho manipular utilitzant el llenguatge C.

Fins ara hem vist que el que abans ens representava un mòdul sencer en VHDL , com el descodificador 7 – Segments (apartat 2.2.3), en C només ens representa una petita funció que ens extreu un valor d'un vector.

Per que respecte a la freqüència de rellotge, veiem com el propi SOPC ens proporciona aquella que nosaltres li indiquem. En aquest cas estem utilitzant la de 125ms degut a que és la que Altera ens dóna com a predeterminada en la *Basic Computer*. Si el NIOS el construïm nosaltres mateixos, podem

predeterminar la freqüència a 1 Hz i ens estalviaríem els petits càlculs quan es causa la interrupció.

Anem construint pas a pas el nostre rellotge digital.

3.4.1. Control del Timer

El nostre rellotge es posarà en funcionament quan nosaltres volem. Per aquesta raó tenim que tindre algun dispositiu que ens pari i ens arranqui el rellotge quan l'iniquem. El Timer ens ofereix la possibilitat de parar-ho i tornar-ho a activar sempre i quan ho indiquem als registres pertinents. Per això utilitzarem un dels interruptors per a modificar els valors dels registres interns del Timer.

Si volem que el Timer corri i ens causi les interrupcions cada segon tenim que activar els següents bits del resgitre *control*: *ITO, CONT* i *START* (veiem la taula de la Fig. 3.20). Pel contrari, si volem que el Timer quedi parat i a l'espera de tornar a ser activat tindrem que activar els bits: CONT i STOP, i apagar els: ITO i START.

En aquest cas serà el SW0 l'encarregat de fer l'activació, com veiem a la taula de la Fig. 3.27 li correspon el primer bit del registre de la memòria base.

Fig. 3.27. Mapa de bits del registre dels interruptors.

Seguint amb el mateix codi que hem utilitzat fins ara, incorporem el següent dintre del programa principal:

```
while(1)
   {
      if ((*(switches) & 0x1)==1)
         *(timer+1)=0x7; // activem el timer i interrupcions
      else
         *(timer+1)=0xA; // desactivem el timer i interrupcions

      hex_segments[0]= descodifica_bcd_7segments (comptador);
      for(i=0; i<8;i++)
         {
            hex_segments[i]=hex_segments[0];
         }
      *(hex)=*(int *)hex_segments;
      *(hex_2)=*(int *)(hex_segments+4);
   };
```

Amb aquest condicional: *if ((*(switches) & 0x1)==1)* estem verificant se el valor del interruptor és actiu o no actiu. Segons el seu valor activem o desactivem el Timer.

Si ho carreguem a la placa veurem que el funcionament és el correcte. Si l'interruptor està activat veiem en els Displays com canvia la xifra cada segon. Si està desactivat el Display queda congelat fins que no es canviï l'estat del interruptor.

3.4.2. Construcció del rellotge

Ara ens ocuparem de crear el rellotge amb els dispositius que ja sabem manipular. Utilitzarem el Timer per a que calculi el temps, l'interruptor per encendre el rellotge i sis displays per visualitzar l'hora a la placa.

Planifiquem el programa, necessitem tres variables que ens donin el valor dels segons, minuts i hores. Aquestes després tindran que ser codificades a 7-segment diferenciant unitats i desenes. Ja tenim una funció que ens fa aquesta última part: *char descodifica_bcd_7segments (int index).* Realitzarem una nova funció que sigui la responsable d'anar incrementant cada segon el rellotge que cridarà la funció interrupció del Timer, aquesta es dirà: *void suma_segons(void).* A més tindrem dos més que s'encarregaran de tornar-nos el valor de les desenes i unitats respectivament del valor que li passem com a paràmetre: *char torna_decenes(int valor)* i *char torna_unitats(int valor)*. Per a que posteriorment es pugui codificar amb només una línia de codi tots els dígits també farem la funció: *void dona_7segments(void),* encarregada de guardar al nostre vector de caràcters els valors del 7 segments.

Aquestes funcions es criden entre elles i finalment obtenim el valor de unitats i desenes de cada un dels enters que *suma_segons()* ens torna.

Veiem el codi de cada una de les funcions:

```
void suma_segons(void)
{
   if (segons < 59)
      segons++;

   else
   {
      segons=0;

      if(minuts < 59)
         minuts++;

      else
      {
```

```
         minuts=0;
         if (hores < 23)
            hores++;
         else
            hores=0;
      }
   }
   dona_7segments();
}

/*****************************************************/
char torna_unitats(int valor)
{
   int unitat;

   unitat= valor%10;
   return descodifica_bcd_7segments(unitat);

}

/*****************************************************/
char torna_decenes(int valor)
{
   int decena;

   decena= valor/10;
   return descodifica_bcd_7segments(decena);
}

/*****************************************************/
void dona_7segments(void)
{
      hex_segments[2]= torna_unitats(segons);
      hex_segments[3]= torna_decenes(segons);
      hex_segments[4]= torna_unitats(minuts);
      hex_segments[5]= torna_decenes(minuts);
      hex_segments[6]= torna_unitats(hores);
      hex_segments[7]= torna_decenes(hores);
}
```

Veiem com l'última funció ja ens introdueix els valors de cada un dels displays dintre del vector corresponent, és després al programa principal, dintre del *while*, que s'iguala a la sortida:

```
while(1)
   {
      if ((*(switches) & 0x1)==1)
            *(timer+1)=0x7;
      else
         *(timer+1)=0xA;

      *(hex)=*(int *)hex_segments;
      *(hex_2)=*(int *)(hex_segments+4);
   };
```

Finalment si compilem aquest fitxer i el carreguem a la placa amb *l'Altera Monitor Program* veiem el resultat de la següent figura:

Fig. 3.28. Visualització final del projecte.

3.4.3. Posem hora

Com ja hem fet en VHDL, arribats a aquest punt volem poder manipular el nostre rellotge per posar-ho en hora. Com en el cas anterior, farem que prement el KEY3 augmentin les hores, i si ho fem al KEY2 ho facin els minuts.

Pushbuttons

Fent servir la mateixa tècnica que hem fet servir fins ara utilitzarem les interrupcions. Així consultem el manual d'Altera [7] per veure el mapa de bits dels botons:

Address	31 30 · · · 4	3	2	1	0	
0x10000050	Unused	KEY_{3-1}				Data register
Unused	Unused					
0x10000058	Unused	Mask bits				Interruptmask register
0x1000005C	Unused	Edge bits				Edgecapture register

Fig. 3.29. Mapa de bits dels botons.

Veiem que tenim dos registres que ens donaran informació de les interrupcions, el *Interruptmask*, encarregat d'habilitar les interrupcions de cada un dels botons i el *Edgecapture*, que ens dona informació de quin dels botons la ha causat.

Altera es reserva el KEY0 per a el reset de la màquina, així aquest botó no el podem manipular, per aquesta mateixa raó el bit '0' de cada registre ens l'indiquen com inutilitzable.

Necessitem que dos botons tinguin les interrupcions habilitades. També voldrem que mentre que el rellotge estigui en marxa aquest botons quedin inhabilitats donant-li prioritat al funcionament continu del rellotge.

Així al programa principal tindrem que habilitar i inhabilitar les interrupcions dels nostres dispositius segons el valor de entrada del SW0. Veiem com ens quedarà:

```
Void main()
{
   //...declaració de les variables

   __builtin_wrctl(0, 1);//Write 1 into status register

   while(1)
   {
      if ((*(switches) & 0x1)==1)
      {
            __builtin_wrctl(3, 0x1);// (ienable=1)
            *(timer+1)=0x7; //
            *(pushbuttons+2)=0x0;
      }

      else
      {
         __builtin_wrctl(3, 0x2);
         *(timer+1)=0xA;
         *(pushbuttons+2)=0xc;
      }

      *(hex)=*(int *)hex_segments;
      *(hex_2)=*(int *)(hex_segments+4);
   };
   return 0;
}
```

Veiem que si l'interruptor està a '1' habilitem el Timer i les seves interrupcions i si no és així activem les interrupcions dels botons.

Un cop decidides les prioritats de funcionament tenim que incorporar les noves funcions per a que puguem posar l'hora. A partir d'ara la funció interrupció dels botons *(void pushbutton_isr(void))* s'encarregarà de verificar quin dels dos botons és el que l'ha causat, és a dir, llegirà el valor del registre *edgecapture*, tant el bit '3' com el '2'. Segons el resultat es cridarà a la funció *void suma_hora(void)* o *void suma_minut(void).* Veiem els codis:

```
void pushbutton_isr(void)
{
   int * pushbuttons= (int *) PUSHBUTTONS_BASE_ADDRESS;

   if ((*(pushbuttons+3) & 0x8)==8) // llegeig l'edgecapture bit3
      suma_hora();
   else
   {
      if((*(pushbuttons+3) & 0x4)==4)// llegeig l'edgecapture bit2
         suma_minut();
   }
```

```
   *(pushbuttons+3)=0;     //neteja l'edgecapture
   return;
}

/*****************************************************************/

void suma_hora(void)
{
   if (hores < 23)
      hores++;
   else
      hores=0;
   dona_7segments();
}

/*****************************************************************/

void suma_minut(void)
{
   if(minuts < 59)
      minuts++;

   else
      minuts=0;
   dona_7segments();
}
```

Si incorporem aquestes modificacions al nostre codi, compilem i carreguem veurem com el rellotge ja funciona correctament.

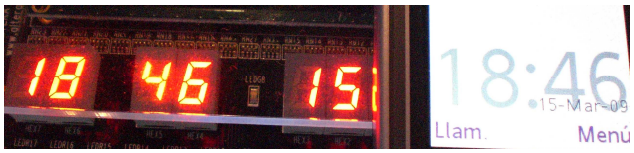


Fig. 3.30. Visualització final del projecte.

3.5. Comparativa: NIOS II i la clàssica FPGA

Com hem comprovat tenim diferents maneres de fer el mateix projecte amb la placa DE2. Altera ens ofereix un estalvi d'energia i capacitat computacional deixant-nos construir el nostre propi microcontrolador NIOS II. Ens estalviem feina i codi ja que podem escollir els diferents dispositius per a que sigui més fàcil i ràpid arribar a un punt final.

Hem vist que en el cas de la FPGA la manera de treballar és més modular. Si necessitem que un dispositiu faci una feina concreta ho tenim que pensar de fer per diferents mòduls, de manera desglossada, pas a pas i sempre amb una mateixa estructura. Tenim que tindre clar les entrades i sortides de cada un dels mòduls i l'arquitectura interna de cada un d'ells té que estar clara.

Per una altra banda tenim la possibilitat que ens dona Altera de crear-nos un microcontrolador amb diferents dispositius que ens millorin la feina. En aquest cas tenim que saber abans de començar el nou projecte com serà, quins dispositius de la placa i interns del NIOS necessitarem i construir-nos en microcontrolador. A partir d'aquí només tenim que tindre clar com el tenim que programar i quin volem que sigui el nostre resultat. Altera ens facilita molt la feina. És veritat que en aquest cas necessitem dominar més d'un programa però no pas més d'un llenguatge de programació.

El que nosaltres fem a mà amb VHDL creant els diferents dispositius com ara el divisor de freqüències, utilitzant el NIOS t'ho genera el propi SOPC Builder quan l'introdueixes el valor del període del Timer que vols utilitzar.

Possibles millores

A partir d'aquest moment tenim les eines i coneixements suficients per continuar avançant amb projectes més complicats. Les possibilitats són infinites. Podem fer des de les aplicacions que hem fet en aquest treball fins a un videojoc utilitzant les entrades i sortides d'àudio i vídeo.

En un àmbit més aplicat al nostre estil de projecte, més enfocat a la docència, podríem fer que alguna de les aplicacions, com ara el multiplexor o el rellotge digital fossin un més dels dispositius que apareixen a la placa, com un Display 7 segments. Que arribada l'hora de crear un nou projecte es poguessin utilitzar com ho hem fet amb qualsevol altre dels que Altera ja ens incorpora a la DE2. Amb una crida a una funció que la mateixa placa reconegués o a una posició de memòria determinada ens podria retornar l'hora en que ens trobem en aquell moment.

També es podrien utilitzar nous programes com ara el NIOS II IDE, pensat per a projectes més professionals i amb uns objectius més comercials.

Conclusions

Finalitzat aquest projecte veiem que tenim moltes possibilitats amb el NIOS II. Tenim, en una petita placa, tota la capacitat que podem trobar en un PC. El podem manipular i programar, fer aplicacions diferents. Tant si la programem amb VHDL com amb C té les seves avantatges i inconvenients.

El que més diferència una de l'altre és la utilització de la capacitat de la placa, on estalviem més construint el nostre propi NIOS. Referent a la programació, utilitzem mètodes diferents, en cas de la programació de la FPGA en VHDL cada cop que fem un nou mòdul pensem les entrades i sortides que intervenen en aquest i les tenim que crear, podem anar incorporant nous dispositius mica en mica sense tindre massa clar des de un inici quin seran aquests. Per una altra banda, si programem el nostre NIOS en C tenim que tindre clar els dispositius que volem utilitzar des d'un principi ja que tenim que crear-los. En canvi a l'hora de construir el codi és molt més ràpid ja que el propi Quartus t'ha creat el codi d'aquest dispositius que amb la FPGA et tens que crear tu mateix.

El fabricant, Altera, ens proporciona tot el material necessari per utilitzar aquesta placa. De vegades, aquest material és excessiu i no massa resolutiu tot i que sí hi ha que és útil.

Per una altra banda hem vist que no és una placa que s'utilitzi molt, costa molt trobar informació i ajuda fora de la pàgina web d'Altera. A més a l'àmbit espanyol podem dir que pràcticament és una desconeguda. Tota la informació al respecte la trobem en anglès.

Bibliografia i referències consultades

[1] Pàgina web d'Altera: http://www.altera.com/education/univ/materials/boards/unv-de2-board.html. Des d'aquesta web podem demanar al fabricant llicència pel nostre programari així com tots els manuals disponibles sobre la nostra placa.

[2] Pàgina web de SED: http://epsc.upc.edu/projectes/sed. En aquesta web podem trobar exemples i apunts per realitzar els nostres exercicis.

[3] PDF: Tutoria sobre el funcionament del SOPC Builder: *Introduction to the Altera SOPC Builder Using VHDL Design* ftp://ftp.**altera**.com/up/pub/Tutorials/DE2/Computer_Organization/tut_**sopc**_introduction_vhdl.pdf

[4] PDF: Introducció al funcionament del NIOS II: *Introduction to the Altera Nios II Soft Processor* www.cs.duke.edu/courses/spring09/cps104/Altera/tut_nios2_introduction.pdf

[5] PDF: Set d'intruccions en assemblador per el NIOS II: *Instruction Set Reference* www.altera.com/literature/hb/nios2/n2cpu_nii51017.pdf

[6] Pàgina web d'Altera on podem extreure la tutoria sobre el funcionament del Monitor Altera Program: http://www.altera.com/education/univ/software/monitor/unv-monitor.html

[7] PDF: Manual de la Màquina Bàsica per NIOS II d'Altera: *Basic Computer System for Altera DE2 Board:* http://www.altera.com/education/univ/support/examples/unv-example-systems.html

[8] PDF: Manual per aprendre a fer simulacions de les interrupcions: *NIOS II Interruption simulation:* lap2.epfl.ch/courses/archord2/labs/interrupt_simulation.pdf

[9] PDF: Datasheet dels registres d'entrada i sortida de NIOS II: *NIOS PIO:* www.altera.com/literature/ds/ds_nios_pio.pdf

[10] PDF: Datasheet del TIMER intern del NIOS II: *NIOS Timer:* www.altera.com/literature/ds/ds_nios_timer.pdf

[11] PDF: Manual del TIMER intern del NIOS II: *Interval Timer Core:* www.altera.com/literature/hb/nios2/n2cpu_nii51008.pdf

[12] PDF: Manual d'introducció al funcionament de la placa DE2: *Getting Starting Guide:* users.ece.gatech.edu/~hamblen/DE2/DE2%20Reference%20Manual.pdf

[13] Kernighan, Brian W., Ritchie, Dennis M., *El Lenguaje de Programación C*, Prentice Hall, Mexico (1991).